Questions with Answer

OBJECT ORIENTED CONCEPTS WITH JAVA [15CS45]

[As per Choice Based Credit System (CBCS) scheme]

(Effective from the academic year 2015 -2016)

SEMESTER - IV

Prepared by:

Mr. Muneshwara M.S
Assistant Professor,
Department of Computer Science & Engineering
BMS Institute of Technology & Management.
Yelahanka, Bangalore-560064

VISION AND MISSION OF THE CS&E DEPARTMENT

Vision
To develop technical professionals acquainted with recent trends and technologies of computer science to serve as valuable resource for the nation/society.

Mission:
Facilitating and exposing the students to various learning opportunities through dedicated academic teaching, guidance and monitoring.

VISION AND MISSION OF THE INSTITUTE

Vision
To emerge as one of the finest technical institutions of higher learning, to develop engineering professionals who are technically competent, ethical and environment friendly for betterment of the society.

Mission
Accomplish stimulating learning environment through high quality academic instruction, innovation and industry-institute interface
OBJECT ORIENTED CONCEPTS

[As per Choice Based Credit System (CBCS) scheme]
(Effective from the academic year 2016 -2017)

SEMESTER – IV

Subject Code 15CS45
Number of Lecture Hours/Week 04
Total Number of Lecture Hours 50

IA Marks 20
Exam Marks 80

Exam Hours 03

CREDITS – 04

MODULE 1:

INTRODUCTION TO OBJECT ORIENTED CONCEPTS

1. Explain various features of object oriented programming. (08 Marks)

Answer:

The various features of object oriented programming are as follows:

Encapsulation: It is the mechanism that binds together code and the data it manipulates, and keeps both safe from outside interference and misuse.
- It is an important principle that supports the object oriented programming.
- The code and data are combined in such a way that that a self-contained “black box” is created, which is an object in this fashion.
- An object is the device that supports encapsulation.

Polymorphism: It is the attribute that allows one interface to control access to a general class of actions.
- It is characterized by the phrase “one interface, multiple methods”.
- The specific action selected is determined by the exact nature of the situation.
- A real world example of polymorphism is a thermostat, the same principle can be applied in programming where we define push( ) and pop( ) while implementing three different types of stacks used for integer values, for character values and for floating-point values.
- It reduces complexity by allowing the same interface to be used to access a general class of actions.

Inheritance: It is the process by which one object can acquire the properties of another object.
- It is important because it supports the concept of classification, where each object would not have to define explicitly all of its characteristics.
- An object needs to define only those qualities that make it unique within its class.
- It is the inheritance that makes it possible for one object to be a specific instance of a more general case which is an important aspect of object-oriented programming.
2. What is inline function? Explain with an example. Mention its advantages. (06 Marks)

Answer:

Inline functions:
These are the functions which are not actually called; rather, their code is expanded in line at the point of each invocation.
- The process is similar to using a function-like macro.
- The definition of the function should be preceded with the ‘inline’ keyword in order for it to be an inline function.

For example:
The following example illustrates the use of inline functions where the function max( ) is expanded instead in line instead of called:

```cpp
#include<iostream>
Using namespace std;
Inline int max( int a, int b)
{
    return a>b ? a : b ;
}
int main()
{
    cout << max (10,20);
cout << “ “ << max ( 99, 98 );
return 0;
}
```

The advantages of inline functions are as follows:
- Inline functions are important because they allow us to create very efficient code.
- Expanding function calls in line can produce faster run times.
- The significant amount of overhead which is generated by the calling and return mechanism can be reduced.
- Arguments being pushed on to the stack, registers being saved when a function is called and then restoring when the function is called are the operations which can be eliminated with inline functions.
3. What is function overloading? Write a program to swap two integers and two float numbers using function overloading. (06 Marks)

Answer:

Function overloading is a way to use the same function name to create functions that perform a variety of different tasks. A family of functions with one function name but with different argument list can be designed.

The program is as follows:

```c
#include<iostream.h>
#include<iomanip.h>
#include<conio.h>

void swap(int *a, int *b)
{
    int c;
    c=*a;
    *a=*b;
    *b=c;
}

void swap(float *a, float *b)
{
    float c;
    c=*a;
    *a=*b;
    *b=c;
}

void swap(double *a, double *b)
{
    double c;
    c=*a;
    *a=*b;
    *b=c;
}

void main()
{
    int a, b;
    float c, d;
    double e, f;
    clrscr();
    cout<<"enter the values of a, b for integer";
    cin>>a>>b;
    cout<<"enter the values for c, d ";
    cin>>c>>d;
    cout<<"enter the values for e, f ";
```
cin>>e>>f;
void swap(int *,int *);
void swap(float *,float *);
void swap(double *,double *);
swap(&a,&b);
swap(&c,&d);
swap(&e,&f);
cout<<"\nafter swapping ";
cout<<"\nvalues of a,b "<<a<<"\t"<<b;
cout<<"\nvalues of c,d "<<c<<"\t"<<d;
cout<<"\nvalues of e,f "<<e<<"\t"<<f;
getch();
}

4. What is a statement? Explain jump statements with syntax. (08 Marks)

**Answer:**

- A statement is a part of the program that can be executed.
- A statement specifies an action.

The jump statements are as follows:

i) **The return statement:** It is used to return from a function.
   - It causes execution to return (jump back) to the point at which the call to the function was made.
   - If return has a value associated with it, that value becomes the return value of the function.
   - Syntax: `return expression;`
   - The expression is present only if the function is declared as returning a value.

ii) **The goto statement:** The goto statement requires a label for operation. The label must be in the same function as the goto that uses it—we cannot jump between functions.
    Syntax: `goto label;`
    ..
    ..
    label:
    where label is any valid label either before or after goto.
Example:

```c
x =1;
loop1:
    x++; 
    if (x < 100) goto loop1;
```

iii) **The break statement:** The break statement has two uses:
    We can use it to terminate a case in the switch statement.
    We can use it to force immediate termination of a loop, by passing the normal conditional test.

When the break statement is encountered inside a loop, the loop is immediately terminated and program control resumes at the next statement following the loop.

iv) The continue statement: Continue forces the next iteration of the loop to take place, skipping any code in between.

For the for loop, continue causes the conditional test and increment portions of the loop to execute.

For the while and do-while loops, program control passes to the conditional tests.

The following program shows the use of continue statement:

```c
void code ( void )
{
    char done, ch;
    done = 0;
    while (! done) {
        ch = getchar ( );
        if ( ch = = 'S') {
            done = 1;
            continue;
        }
        putchar ( ch + 1);
    }
}
```

5. What is inline function? Write a C++ function to find maximum of 2 numbers using inline. (04 Marks)

Answer:

a) Inline function: These are the functions which are not actually called; rather, their code is expanded in line at the point of each invocation.

- The process is similar to using a function-like macro.
- The definition of the function should be preceded with the ‘inline’ keyword in order for it to be an inline function.

The C++ function to find maximum of two numbers is as follows:

```c
#include<iostream>
using namespace std;
inline int min(int a, int b)
{
    return a < b ? a :b ;
}
int main ( )
{
    cout << min (10, 20);
    return 0;
}
```
6. **What is function overloading? Explain with example, why function overloading is important?** (08 Marks)

**Answer:**

**Function overloading:**

It is the process of using the same name for two or more functions. Each redefinition of the function must use either different types of parameters or a different number of parameters.

- Function overloading is important because it provides support for compile-time polymorphism, it also adds flexibility and convenience.
- Using the concept of function overloading, we can design a family of functions with one function name but with different argument lists.
- The function would perform different operations depending on argument list in the function call.
- The correct function to be invoked is determined by checking the number and type of the arguments but not the function type.

The following program illustrates the function overloading:

```cpp
#include<iostream>
using namespace std;
int myfunc (int i);
double myfunc(double i);
int main ( )
{
    cout << myfunc (10) << " ", // calls myfunc (int i)
    cout << myfunc ( 5.4 );  // calls myfunc (double i)
    return 0;
}
double myfunc ( double i )
{
    return i;
}
int myfunc ( int i)
{
    return i;
}
```

The program demonstrates how function overloading can be used to define multiple functions with the same name, each accepting different types of arguments. This allows for more flexibility and convenience in programming, as it allows for the same function name to be used in different contexts with different types of data.
7. Explain the following OOP features:
   i) Class     ii) Encapsulation    iii) Polymorphism    iv) Inheritance    (08 Marks)

   Answer:
   i) Class: Classes are created using the key word class.
       - A class declaration defines a new type that links code and data.
       - This new type is then used to declare objects of that class.
       - Class is a logical abstraction, but an object has physical existence, where object is an
         instance of a class.
       - The general class declaration is as follows:
         
         ```
         class class-name {
         private data and functions
         access specifier: data and functions
         } object list;
         ```
       
   [REST OF THEM REPEATED]

   8. Define inline functions. Example with an example program. What are the conditions,
      where inline functions cannot be expanded?

   Answer:
   [REPEATED]
   CONTINUATION OF THAT:

   The conditions where inline functions cannot be expanded are:
   1) For functions returning values, if a loop, a switch or goto exists.
   2) For functions not returning values, if a return statement exists.
   3) If functions contain static variables.
   4) If inline functions are recursive.

   9. Define function overloading. Demonstrate with C++ program.

   Answer:

   Function overloading: It is the process of using the same name for two or more functions. Each
   redefinition of the function must use either different types of parameters or a different number of
   parameters.

   The following program illustrates the function overloading:
   ```
   #include <iostream>
   using namespace std;
   int myfunc (int i);
   double myfunc (double i);
   int main ()
   {
   cout << myfunc (10) << " ";
   // calls myfunc ( int i )
   cout << myfunc (5.4);
   // calls myfunc ( double i )
   return 0;
   } 
   ```

   Prepared by: Mr. Muneshwara M.S, Asst. Prof., Dept. of CS&E, BMSIT&M, Yelahanka, Bengaluru -64
double myfunc ( double i )
{
    return i;
}
int myfunc ( int i )
{
    return i;
}

10. What is a class? Write a program to create a class called employee which contains name, desg, ecode and salary as data member and read, write as function member, using this class to read and print 10 employee information.

**Solution:**

**Class:** Classes are created using the key word class.
- A class declaration defines a new type that links code and data.
- This new type is then used to declare objects of that class.
- Class is a logical abstraction, but an object has physical existence.

The program is as follows:
```cpp
class employee
{
    private: char name [20];
        int desg, ecode, sal;
    public: void read ( char *n, int a, int s);
            void write ( );
};
void employee : : read (char *n, char d, int e, int s)
{
    strcpy(name,b);
    desg = d;
    ecode = e;
    sal = s;
}
void employee : : write ( )
{
    cout << “employee.name” << name;
    cout << “employee.desg” << desg;
    cout << “employee.ecode” << ecode;
    cout << “employee.sal” << sal;
}
main ()
{
    employee e[10];
    char ename, edesg;
```
int ecode, esal, i;
for(i=0 ; i < 10; i++)
{
    cout << "enter employee name, designation, ecode and salary";
    cin >> ename >> edesg >> ecode >> esal;
    e[i].read(ename, edesg, ecode, esal);
}
cout << "employees information";
for (i=0 ; i<10 ; i++)
{
    e[i].write();
}

11. What is constructor? Mention its types. Explain the parameterized constructor with an example. (06 Marks)

Answer:

Constructor: A constructor is a special function that is a member of a class and has the same name as that of that class.
The types of constructors are as follows:
   a) Parameterized constructors
   b) Copy constructors
   c) Dynamic constructors.

Parameterized constructor:
The constructor that can take arguments are called Parameterized constructors.
The following example illustrates the parameterized constructors:

```cpp
#include<iostream.h>
class box {
    int length, breadth, height;
public:
    int volume (int length, int breadth, int height);
    box ();
};
box : : box ()
{
    length = 0;
    breadth =0;
    height =0;
}
int box : : volume (int length, int breadth, int height)
{
    int vol;
    vol = length * breadth * height;
    return vol;
}
```
int main ()
{
    box b;
    cout << “the volume is: “;
    int a = b.volume (5,8,10);
    cout << a;
}

12. Explain three access specifiers. (06 Marks)

Answer:

The three access specifiers are as follows:

i)public: The public access specifier allows functions or data to be accessible by other parts of the program.
   - Public members can be accessed from outside the class also.

ii)protected: This access specifier is needed only when inheritance is involved.

iii)private: The private access specifier allows data to be used by that particular function in which they are declared and initialized.
   - If both the labels are missing, then, by default, all the members are private.

13. What is class? Explain the syntax of class. (08 Marks)

Answer:

Classes are created using the key word class.

- A class declaration defines a new type that links code and data.
- This new type is then used to declare objects of that class.
- Class is a logical abstraction, but an object has physical existence, where object is an instance of a class.
- The general class declaration is as follows:

```
class class-name {
    private data and functions
    access specifier: data and functions
    access specifier: data functions
    // ....
} object list;
```

The object list is optional. If present, it declares objects of the class.

Access-specifier is one of the following keywords:

i)public: The public access specifier allows functions or data to be accessible by other parts of the program.

ii)protected: This access specifier is needed only when inheritance is involved.
iii) **Private**: The private access specifier allows data to be used by that particular function in which they are declared and initialized.

For example:
```cpp
class simple {
private:
    int a;
    int b;
public:
    float c;
    void set_data (int x; int y; )
    {
        a=x;
        b=y;
    }
    int get_data ( )
    {
        return ( a + b );
    }
};
```

Here, `simple` is defined as a class by using “class” keyword which has two private integers, one public integer and two public functions.

14. **Mention the restrictions that are placed on static member functions.** (04 Marks)

**Answer:**

The restrictions that are placed on static member functions are:

i) They may only directly refer to other static members of the class.

ii) A static member function does not have a this pointer.

iii) There cannot be a static and a non-static version of the same function.

iv) A static member function may not be virtual.

v) They cannot be declared as const or volatile.

15. **What is parameterized constructor? Explain the different methods of passing arguments to the parameterized constructor with example.** (08 Marks)

**Answer:**

The constructor that can take arguments are called parameterized constructors.

Arguments can be passed to parameterized constructor in two ways as follows:

i) By calling the constructor explicitly

ii) By calling the constructor implicitly.

The explicit call can be illustrated as follows:
```cpp
    integer int1 = integer (0,100);
```
This statement creates an integer object `int1` and passes the values of 0 and 100 to it.

The implicit call can be illustrated as follows:
integer int1(0, 100);
This method is sometimes called as shorthand method, which is used very often as it is shorter, looks better and is easy to implement.
The following program shows both the explicit and implicit call:
#include<iostream>
using namespace std;
class integer {
    int m,n;
public:
    integer (int, int);              // constructor declared
    void display (void)
    {
        cout << "m =" << m << "n =" << n;
    }
};
integer :: integer (int x, int y)  // constructor defined
{
    m=x; n=y;
}
int main ()
{
    integer int1(0, 100);            // constructor called implicitly
    integer int2 = integer (25,75);  // constructor called explicitly
    cout << "no object1" << "n =";
    int1.display ();
    cout << "no object2" << "n =";
    int2.display ();
    return 0;
}

16. Write a C++ program to keep track of the number of objects created by a particular class without using extern variable.

Answer:

template <typename T>
struct counter
{
    counter ( )
    {
        objects_created ++;
        objects_alive++;
    }
    virtual counter ( )
    {
        objects_alive;
    }
static int objects_created;
static int objects_alive;
};
template <typename T> int counter <T> : : objects_created ( 0 );
template <typename T> int counter <T> : : objects_created ( 0 );
class X : counter < X >
{
  //.....
};
class Y : counter < Y >
{
  //.....
};
int main ( )
{
x1;
{
  x2;
  x3;
  x4;
  x5;
  y1;
  y2;
}  
y3;  // objects gone
  cout << “created:”;
  << “X: “ << counter<X> : : object_created
  << “Y: “ << counter<Y> : : object_created
  << endl;
  cout << “alive:”
  << “X: “ << counter<X> : : object_alive
  << “Y: “ << counter<Y> : : object_alive
  << endl;

17. Demonstrate the output for the following snippets and comment.
i) Class A
{
  int x=10;
  void display ( )
  {
    cout << “the value of X = “ << x;
  }
};
A obj;
obj.display ( );
i) class A
```cpp
{
    int pvt
    public: int *ptr_pub;
    A ( ) {
        pvt = 25;
        ptr_pub = &pvt;
    }
    void print_private ( )
    {
        cout << pvt << endl;
    }
    void main ( )
    {
        A obj A;
        *obj A.ptr_pub = 10;
        obj A.print_private ( );
    }
}

Answer:

i) Output:
The value of x = 10.
ii) Output:
25

18. Demonstrate with C++ programs for
i) Passing objects to functions
ii) Returning objects.
(08 Marks)

Answer:
i) Passing objects to functions:
// Passing an object to the function
#include<iostream>
using namespace std;
class myclass {
    int i;
    public:
    myclass ( int n );
    ~myclass ( );
    void set_i ( int n ) { i =n; }
    int get_i ( ) { return i ; }
};
myclass :: myclass ( int n )
{
    i =n;
    cout << “constructing” << i << “n”;
};
```
myclass : : ~ myclass()
{
    cout << “Destroying” << i << “n”; // Destructor
}

void f(myclass ob);

int main()
{
    myclass o(1);
f(o);
    cout << “This is i in main: “;
cout << o.get_i() << “n”;
    return 0;
}

void f(myclass ob)
{
    ob.set_i(2);
    cout << “This is local i: “ << ob.get_i();
    cout << “n”;
}

Output:
Constructing 1
This is local i: 2
Destroying 2
This is i in main: 1
Destroying 1

ii) Returning Objects:

// Returning objects from a function
#include<iostream>
using namespace std;
class myclass {
    int i;
public:
    void set_i(int n) { i=n; }
    int get_i() { return i; }
};
myclass f();

int main()
{
    myclass o;
o = f();
cout << o.get_i() << “n”; // Calling constructor
    return 0;
}

myclass f()
{
    myclass x;
19. Why friend function is required? Write a program to add two complex numbers using friend function.

(08 Marks)

Answer:

The need of friend functions are:
a) Friend functions can be useful when we are overloading certain types of operators.
b) Friend functions make the creation of some types of I/O functions easier.
c) Friend functions may be desirable in cases as two or more classes may contain members that are interrelated relative to other parts of the program.

The program to add two complex numbers using friend functions is as follows:

```cpp
#include<iostream>
using namespace std;
class Complex {
    int real, imagin;
    public:
    void get() {
        cout << "Enter the real part\n";
        cin >> real;
        cout << "Enter the imaginary part\n";
        cin >> imagin;
    }
friend void sum(Complex, Complex);
};
void sum(Complex c1, Complex c2) {
    cout << "result\n";
    cout << c1.real + c2.real << " + i" << c1.imagin + c2.imagin;
}
void main() {
    complex op1, op2;
    cout << "addition of two complex numbers using friend functions\n\n";
    op1.get();
op2.get();
    sum(op1, op2);
}
```
20. Explain generic function with an example. (06 Marks)

Answer:

- A generic function defines a general set of operations that will be applied to various types of data.
- The type of data that the function will operate upon is passed to it as parameter.
- Through a generic function, a single general procedure can be applied to a wide range of data.
- A generic function is created using the keyword template.

The following example creates a generic function that swaps the values of the two variables with which it is called:

```cpp
#include<iostream>
using namespace std;
template <class X> void swapargs (X &a, X &b)
{
    X temp;
    temp = a;
    a = b;
    b = temp;
}
int main ( )
{
    int i=10, j=20;
    double x=10.1, y=23.3;
    char a='x', b='z';
    cout << "Original i and j are:" << i << ' ' << j << 'n';
    cout << "Original x and y are:" << x << ' ' << y << 'n';
    cout << "Original a and b are:" << a << ' ' << b << 'n';
    swapargs(i, j);
    swapargs(x, y);
    swapargs(a, b);
    cout<<"Swapped i and j are" << i << ' ' << j << 'n';
    cout<<"Swapped x and y are" << x << ' ' << y << 'n';
    cout<<"Swapped a and b are" << a << ' ' << b << 'n';
    return 0;
}
```
21. What are friend functions? What are the advantages of using friend functions? Write a C++ program to find sum of two numbers using friend functions. (08 Marks)

Answer:
Friend function: It is a function by which it is possible to grant a nonmember function access to the private members of a class by using it. A friend function has access to all private and protected members of the class for which it is a friend.

The advantages of using friend functions are:
a) Friend functions can be useful when we are overloading certain types of operators.
b) Friend functions make the creation of some types of I/O functions easier.
c) Friend functions may be desirable in cases as two or more classes may contain members that are interrelated relative to other parts of the program.

The program to find sum of two numbers using friend function is as follows:

```cpp
#include<iostream>
using namespace std;

class A {
private:
    int x, y;
public:
    void set_xy ( int i, int j );
    friend int add ( A ob);
    add ( );
};

void A :: set_xy ( int i, int j)
{
    x = i;
    y = j;
}

int add ( A ob)
{
    return ob.x + ob.y;
}

void main ( )
{
    A ob1;
    ob1.set_xy (10, 20);
    cout << add ( ob1);
}
```

MODULE 2.
INTRODUCTION TO JAVA

1. How ‘compile once and run anywhere’ is implemented in Java, Explain. (4M) [July 2014, July 2016]

Answer:

Architecture-Neutral
A central issue for the Java designers was that of code longevity and portability. One of the main problems facing programmers is that no guarantee exists that if you write a program today, it will run tomorrow—even on the same machine. Operating system upgrades, processor upgrades, and changes in core system resources can all combine to make a program malfunction. The Java designers made several hard decisions in the Java language and the Java Virtual Machine in an attempt to alter this situation. Their goal was “write once; run anywhere, any time, forever.” To a great extent, this goal was accomplished.

2. List and explain the Java buzzwords. (6M) [July 2014, July 2016]

Answer:

Sun micro system officially describes java with a list of buzz words or attributes. They
- Simple & powerful
- Safe
- Object oriented
- Robust
- Architecture neutral
- Compiled & Interpreted
- Multithreaded
- Easy to learn

The salient features of Java are as follows:

- Simple & Powerful: To make the language familiar to the existing programming, java is modelled on C & C++. Java empowers you to express every idea you have in a clean object oriented way.
- Safe: Threat of viruses and abuse of resources are everywhere, java system not only verify the memory resources but also ensures that no viruses are communicated with the applet. The absence of pointers in java ensures that program can not gain access to memory location.

- Object-oriented: Java is an object-oriented language. Almost every thing in java is object. All the program codes & data reside within object & classes.

- Robust: Java is a strictly typed language, because the types must match exactly. It checks your code at compile time as well as at run time. Java is a garbage collected language, relieving the programmers all memory management problems (i.e., deallocation is completely automatic).

- Java incorporates exception handling which captures series of errors and eliminates any risk of crashing the system.

- Architecture neutral: Java is the language that is not tied to any particular hardware or operating system. Program developed in java can be executed anywhere on any system. You can “write once, run anywhere, anytime forever”. Changes & upgrades in operating system, processors will not force any changes in java program. It works on Macintosh PC, UNIX & whatever the future platforms can offer.

- Interpreted: Java accomplishes architecture neutrality by compiling the java source code into an intermediate code called “byte codes”, which can be interpreted on any system that has a proper java runtime on it. Multithreaded: Java supports multithreaded programming which allows you to write programs that do many things simultaneously.

- Easy to learn: The language features feel like the natural way to do things & encourage good programming style. Since object model is both mandatory & simple, you will quickly become acquainted with object oriented style of programming.
3. **Describe the Simple Java Program?**

**Answer:**

```java
class Message {
    /**
     * This is a main method.
     * @param args passed to the main method
     */
    public static void main(String[] args) {
        /* Printing the message */
        System.out.println("Welcome to Java World! ");
    }
}
```

- **members**, is done within the open `{ }` and closed `{ }` curly braces. This marks the beginning and end of the class definition block.
- **class** keyword is used to declare a class in java.
- **public** keyword is an access modifier which represents visibility, it means it is visible to all.
- **static** is a keyword, if we declare any method as static, it is known as static method. The core advantage of static method is that there is no need to create object to invoke the static method. The main method is executed by the JVM, so it doesn't require to create object to invoke the main method. So it saves memory.
- **void** is the return type of the method, it means it doesn't return any value.
- **main** represents startup of the program.
- **String[] args** is used for command line argument.
  - **args** is an array which receives the command line argument when the program runs.
- **System.out.println()** is used print statement.

  **System is a Predefined Class in Java**

  - **out is the Output Stream which is related to console**
- **Having semicolon at the end of class in java is optional.**
4. Explain: i) >>> ii) short circuit logical operators iii) for each. (6M) [July 2014]

Answer:
   i) int num, den;
   if (den != 0 && num > 2) {
   }

   ii) int num, den;
   if (den != 0 && num <= 0 || den == 0) {
   }

An object is a single instance of a class that retains the structure and behavior as defined by the class. These objects are sometimes called *instances of a class*

4. Describe the process of building and running Java program. (4M) [July 2014, July 2016]

Answer:

BYTE Code is intermediate level code, which is interpreted by the JVM. It is not directly executable on the machine. This gives Java its “write once and run anywhere” nature.

When Java program is written and compiled then it will create a .class file which consists of byte code instructions, understandable to JVM. This class file is system independent.

Every system has its own JVM. So JVM will convert this byte code into machine language understandable to that system. So it has run write once and run anywhere nature.

Java achieves architecture neutrality in the following way. Being platform independent was one of the major objectives for Java. Java achieves this independence by introducing an intermediate code representation of compiled Java programs. Programs are compiled into a byte code which is then interpreted by platform specific interpreter. The byte code is same for any architecture, IBM compatible, Apple, Sparc, Sun Solaris.

Java program
Java compiler
The Java Virtual Machine, or JVM, is an abstract computer that runs compiled Java programs. The JVM is "virtual" because it is generally implemented in software on top of a "real" hardware platform and operating system. All Java programs are compiled for the JVM. Therefore, the JVM must be implemented on a particular platform before compiled Java programs will run on that platform.

5. **Explain arrays in java with examples** (6M) [July 2016, July 2014, Jan 2015]

   **Answer:**

   Arrays in Java are actual objects that can be passed around and treated just like other objects.

   Arrays are a way to store a list of items. Each slot of the array holds an individual element, and you can place elements into or change the contents of those slots as you need to.

   Three steps to create an array:
   1. Declare a variable to hold the array.
   2. Create a new array object and assign it to the array variable.
   3. Store things in that array.

   E.g.

   ```java
   String[] names;
   names = new String[10];
   names[1] = "n1";
   names[2] = 'n2';
   ```

6. **What is jump statement?** (4M) [July 2014, July 15]

   **Answer:**

   - The if conditional, which enables you to execute different bits of code based on a simple test in Java, is nearly identical to if statements in C.
   - if conditionals contain the keyword if, followed by a boolean test, followed by a statement (often a block statement) to execute if the test is true:
   - if (x < y)
     System.out.println("x is smaller than y");
An optional else keyword provides the statement to execute if the test is false: if 
(x < y) 
System.out.println("x is smaller than y"); else 
System.out.println("y is bigger");

7. Discuss break and continue (5M) [Jan 2015,Jan 2016]

**Answer:**

An alternative to using the if and else keywords in a conditional statement is to use the conditional operator, sometimes called the ternary operator.

The *conditional operator* is a *ternary operator* because it has three terms.

Syntax : test ? trueresult : falseresult

The *test* is an expression that returns true or false, just like the test in the if statement. If the test is true, the conditional operator returns the value of *trueresult*; if it's false, it returns the value of *falseresult*. For example, the following conditional tests the values of x and y, returns the smaller of the two, and assigns that value to the variable smaller: int smaller = x < y ? x : y;

The conditional operator has a very low precedence; that is, it's usually evaluated only after all its subexpressions are evaluated. The only operators lower in precedence are the assignment operators.

8. Explain about JDK (7M) [July 2014,July 15,Jan 2016]

**Answer:**

The Java Virtual Machine, or JVM, is an abstract computer that runs compiled Java programs. The JVM is "virtual" because it is generally implemented in software on top of a "real" hardware platform and operating system. All Java programs are compiled for the JVM. Therefore, the JVM must be implemented on a particular platform before compiled Java programs will run on that platform.

Java achieves architecture neutrality in the following way. Being platform independent was one of the major objectives for java. Java achieves this independence by introducing an intermediate code representation of compiled java programs. Programs are compiled into a
byte code which is then interpreted by platform specific interpreter. The byte code is same for any architecture, IBM compatible, Apple, Sparc, Sun Solaris.

Java program
Java compiler

9. Discuss three OOP principles (6M) [Jan 2015, July 15]

Answer:
In java basis of encapsulation is a class. You create a class that represents an abstraction for a set of objects that share the same structure and behavior. An object is a single instance of a class that retains the structure and behavior as defined by the class. These objects are some times called instances of a class. The individual or data representation of a class is defined by a set of instance variables. These variables hold the dynamic state of each instance of a class. The behavior of a class is defined by methods that operate on that instance data. A method is a message to take some action on an object. Since the goal is to encapsulate complexity, there are mechanisms for hiding the data declared inside class. Each method or variable in a class may be marked private or public. You can declare private methods and instance data that can not be accessed by any other code outside the implementation of your class.

Inheritance: Inheritance is the process by which object of one class acquire the properties of objects of another class. Inheritance supports the concept of hierarchical classification. For example, the bird robin is a part of the class flying bird, which is again part of a class bird as illustrated in figure below

The principle behind this sort of division is that each derived class shares common characteristics with the class from which it is derived. In OOP, the concept of inheritance provides the idea of reusability. The new class will have the combined features of both the classes. Java is said to be single inheritance language. Multiple inheritance which is explicitly not a feature of java

Polymorphism: Polymorphism means ability take more than one form or polymorphism means one object, many shapes, a simple concept that allows a method to have multiple implementations that are selected based on the number & type of the arguments passed into the method invocation. This is known as method overloading. Figure illustrate the method overloading

Listed below are some major C++ features that were intentionally omitted from java
· Java does not support global variables. It is impossible to create a global variable that is outside of all the classes.
· Java has no goto statement
· Java does not use pointers or addresses in memory, because pointers are unsafe. Improper pointer arithmetic causes most of the bugs in today’s code.

10. **Difference between C++ and Java**

**Answer:**

<table>
<thead>
<tr>
<th>Aspects</th>
<th>C++</th>
<th>Java</th>
</tr>
</thead>
<tbody>
<tr>
<td>Developed Year</td>
<td>1979</td>
<td>1991</td>
</tr>
<tr>
<td>Developed By</td>
<td>Bjarne Stroustrup</td>
<td>James Gosling</td>
</tr>
<tr>
<td>Successor of</td>
<td>C</td>
<td>C(Syntax) &amp; C++ (Structure)</td>
</tr>
<tr>
<td>Paradigms</td>
<td>Object Oriented</td>
<td>TRUE Object Oriented</td>
</tr>
<tr>
<td>Platform Dependency</td>
<td>Dependent</td>
<td>Independent</td>
</tr>
<tr>
<td>Keywords</td>
<td>63</td>
<td>50 defined (goto, const unusable)</td>
</tr>
<tr>
<td>Datatypes : union, structure</td>
<td>Supported</td>
<td>Not Supported</td>
</tr>
<tr>
<td>Pre-processor directives</td>
<td>Supported (#include, #define)</td>
<td>Not Supported</td>
</tr>
<tr>
<td>Header files</td>
<td>Supported</td>
<td>Use Packages (import)</td>
</tr>
<tr>
<td>Inheritance</td>
<td>Supported</td>
<td>Multiple Inheritance not Supported</td>
</tr>
<tr>
<td>Overloading</td>
<td>Supported</td>
<td>Operator Overloading not Supported</td>
</tr>
<tr>
<td>Pointers</td>
<td>Supported</td>
<td>No Pointers</td>
</tr>
<tr>
<td>Code Translation</td>
<td>Compiled</td>
<td>Interpreted</td>
</tr>
<tr>
<td>Storage Allocation</td>
<td>Uses new, delete</td>
<td>uses garbage collector</td>
</tr>
<tr>
<td>Multi-threading and Interfaces</td>
<td>Not Supported</td>
<td>Supported</td>
</tr>
<tr>
<td>Feature</td>
<td>Supported</td>
<td>Not Supported</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>------------</td>
<td>---------------</td>
</tr>
<tr>
<td>Exception Handling</td>
<td>Supported</td>
<td>Supported</td>
</tr>
<tr>
<td>Templates</td>
<td>Supported</td>
<td>Not Supported</td>
</tr>
<tr>
<td>Storage class: auto, extern</td>
<td>Supported</td>
<td>Not Supported</td>
</tr>
<tr>
<td>Destructors</td>
<td>Supported</td>
<td>Not Supported</td>
</tr>
<tr>
<td>Database Connectivity</td>
<td>Not Supported</td>
<td>Supported</td>
</tr>
</tbody>
</table>
MODULE 3.

CLASSES, INHERITANCE, EXCEPTIONS, PACKAGES AND INTERFACES:

1. Describe the java classes?
   
   **Answer:**
   - A Class defines an entity in terms of common characteristics and actions.
   - Class is a mechanism used to group properties of actions common to various objects.
   - “A class is a blueprint for a group of objects that have common properties and behavior.”

   ![Diagram of classes and objects]

   **Syntax**

   ```java
   class <classname>
   {
       <body of the class>
   }
   ```

   *where,*

   class is the keyword used for creating a class,
   <classname> is the name of the class, and
   <body of the class> consists of declaration of attributes and methods.
2. Describe the significance of final and super, with examples. (6M) [Jan 2014, Jan 2015, Jan 2016]

Answer:
When Sun was designing Java, it omitted multiple inheritance - or more precisely multiple implementation inheritance - on purpose. Yet multiple inheritance can be useful, particularly when the potential ancestors of a class have orthogonal concerns. This article presents a utility class that not only allows multiple inheritance to be simulated, but also has other far-reaching applications.

Here, Person is a concrete class that represents a person, while Employment is another concrete class that represents the details of a person who is employed. If you could only put them together, you would have everything necessary to define and implement an Employee class. Except in Java - you can't. Inheriting implementation from more than one superclass - multiple implementation inheritance - is not a feature of the language. Java allows a class to have a single superclass and no more.

3. Explain the inheritance in Java?
Answer:
- Inheritance in java is a mechanism in which one object acquires all the properties and behaviors of parent object.
- The idea behind inheritance in java is that you can create new classes that are built upon existing classes.
- When you inherit from an existing class, you can reuse methods and fields of parent class, and you can add new methods and fields also.
- Inheritance represents the IS-A relationship, also known as parent-child relationship.
- Uses of inheritance in java
- For Method Overriding (so runtime polymorphism can be achieved).
- For Code Reusability.

Syntax of Java Inheritance

class Subclass-name extends Superclass-name
{
    //methods and fields
The extends keyword indicates that you are making a new class that derives from an existing class. The meaning of "extends" is to increase the functionality. In the terminology of Java, a class which is inherited is called parent or super class and the new class is called child or subclass.

4. Explain the types of inheritance?
Answer:
- On the basis of class, there can be three types of inheritance in java: single, multilevel and hierarchical.
- In java programming, multiple and hybrid inheritance is supported through interface only.

Multiple inheritance is not supported in java
- To reduce the complexity and simplify the language, multiple inheritance is not supported in java.
- Consider a scenario where A, B and C are three classes. The C class inherits A and B classes. If A and B classes have same method and you call it from child class object, there will be ambiguity to call method of A or B class.

//SINGLE INHERITANCE

class Animal
{
    void eat()
    {
        System.out.println("eating...");
    }
}
class Dog extends Animal
{
    void bark()
class TestInheritancesingle
{
    public static void main(String args[])
    {
        Dog d=new Dog();
        d.bark();
        d.eat();
    }
}

5. Explain the multi level hierarchy in Java?

Answer :

//MULTILEVEL INHERITANCE

class Animal
{
    void eat()
    {
        System.out.println("eating...");
    }
}
class Dog extends Animal
{
    void bark()
    {
        System.out.println("barking...");
    }
}
class BabyDog extends Dog
{
    void weep()
    {
        System.out.println("weeping...");
    }
}
class TestInheritance2multilevel
{
    public static void main(String args[])
    {
        BabyDog d=new BabyDog();
        d.weep();
        d.bark();
        d.eat();
    }
}
6. Explain the Java Garbage collector?

Answer:
- In java, garbage means unreferenced objects.
- Garbage Collection is process of reclaiming the runtime unused memory automatically.
  
  In other words, it is a way to destroy the unused objects.
- Object is Created Using `new` Operator.
- Object gets memory inside “Heap“.
  
  In C++ After Using Object , Memory for Object gets de-allocated using `delete()`.
  
  In Java De-allocation of Object Can be done automatically.
- Automatic Technique used in Java Programming Which is used to de-allocate memory is called as “Garbage Collection“.
- Java is Smart Enough to identify the Unused objects or Useless Objects.
- we were using `free()` function in C language and `delete()` in C++.
- But, in java it is performed automatically. So, java provides better memory management.
- Garbage Collection is Done `Automatically by JVM`.
- As soon as compiler detects that – Object is no longer needed inside program.
- Garbage Collection Algorithm gets executed automatically to free up memory from the heap so that free memory may be used by other objects.
- Different Java Run times may have different approaches for Garbage Collection

**Advantage of Garbage Collection**
- It makes java memory efficient because garbage collector removes the unreferenced objects from heap memory.
- It is automatically done by the garbage collector(a part of JVM) so we don't need to make extra efforts.

```java
public class TestGarbage1
{
    public void finalize()
    {
        System.out.println("object is garbage collected");
    }
    public static void main(String args[])
    {
        TestGarbage1 s1=new TestGarbage1();
        TestGarbage1 s2=new TestGarbage1();
        s1=null;
        s2=null;
        System.gc();
    }
}
```
7. Write short notes on finalize() method and gc() method

Answer:

finalize() method
- The finalize() method is invoked each time before the object is garbage collected. This method can be used to perform cleanup processing. This method is defined in Object class as:

```java
protected void finalize()
{
}
```

gc() method
- The gc() method is used to invoke the garbage collector to perform cleanup processing. The gc() is found in System and Runtime classes.

```java
public static void gc()
{
}
```

8. Why overriding methods are used in java? Example(8M) [July 2016, July 2015]

Overriding Methods

Answer:
- When a method is called on an object, Java looks for that method definition in the class of that object, and if it doesn't find one, it passes the method call up the class hierarchy until a method definition is found.
- Method inheritance enables you to define and use methods repeatedly in subclasses without having to duplicate the code itself.
- However, there may be times when you want an object to respond to the same methods but have different behavior when that method is called. In this case, you can override that method. Overriding a method involves defining a method in a subclass that has the same signature as a method in a superclass. Then, when that method is called, the method in the subclass is found and executed instead of the one in the superclass.
9. What is meant by instance variable hiding? How to overcome it?

```java
public static void main (String args[]) { Motorcycle m = new Motorcycle(); m.make = "Yamaha RZ350";
   (4M) [July 2014,Jan 2016]

   Answer:
   m.color = "yellow";
   System.out.println("Calling showAtts...");
   m.showAtts(); System.out.println("--------");
   System.out.println("Starting engine..."); m.startEngine(); System.out.println("--------");
   System.out.println("Calling showAtts...");
   m.showAtts();
   System.out.println("--------");
   System.out.println("Starting engine...");
   m.startEngine();
}
```

With the main() method, the Motorcycle class is now an application, and you can compile it again and this time it'll run. Here's how the output should look:

Calling showAtts...
This motorcycle is a yellow Yamaha RZ350 The engine is off. Starting engine... The engine is now on.
Calling showAtts...
This motorcycle is a yellow Yamaha RZ350 The engine is on.
Starting engine...
The engine is already on.
10. Explain constructor method: how it differs from other member function(5M)

[Jan 2014, July 2014]

Answer:
The program looks like this now:
```java
class Motorcycle {
    String make;
    String color;
    boolean engineState;

    void startEngine() {
        if (engineState == true)
            System.out.println("The engine is on.");
        else
            System.out.println("The engine is off.");
    }
}
```
The `showAtts` method prints the current values of the instance variables in an instance of your `Motorcycle` class. Here's what it looks like:
```java
void showAtts() {
    System.out.println("This motorcycle is a " + color + " " + make);
    if (engineState == true)
        System.out.println("The engine is on.");
    else
        System.out.println("The engine is off.");
}
```
The `showAtts` method prints two lines to the screen: the make and color of the motorcycle object, and whether or not the engine is on or off.

11. Difference between method overloading & overriding(6M) [July 2015]

Answer:

Overriding Methods
- When a method is called on an object, Java looks for that method definition in the class of that object, and if it doesn't find one, it passes the method call up the class hierarchy until a method definition is found.
- Method inheritance enables you to define and use methods repeatedly in subclasses without having to duplicate the code itself.

**overriding**

However, there may be times when you want an object to respond to the same methods but have different behavior when that method is called. In this case, you can override that method. Overriding a method involves defining a method in a subclass that has the same signature as a method in a superclass. Then, when that method is called, the method in the subclass is found and executed instead of the one in the superclass.

<table>
<thead>
<tr>
<th>No.</th>
<th>Method Overloading</th>
<th>Method Overriding</th>
</tr>
</thead>
<tbody>
<tr>
<td>1)</td>
<td>Method overloading is used to <em>increase the readability</em> of the program.</td>
<td>Method overloading is used to <em>provide the specific implementation</em> of the method that is already provided by its super class.</td>
</tr>
<tr>
<td>2)</td>
<td>Method overloading is performed <em>within class</em>.</td>
<td>Method overloading occurs <em>in two classes</em> that have IS-A (inheritance) relationship.</td>
</tr>
<tr>
<td>3)</td>
<td>In case of method overloading, <em>parameter must be different</em>.</td>
<td>In case of method overriding, <em>parameter must be same</em>.</td>
</tr>
<tr>
<td>4)</td>
<td>Method overloading is the example of <em>compile time polymorphism</em>.</td>
<td>Method overloading is the example of <em>run time polymorphism</em>.</td>
</tr>
<tr>
<td>5)</td>
<td>In java, method overloading can't be performed by changing return type of the method only. <em>Return type can be same or different</em> in method overloading. But you must have to change the parameter.</td>
<td><em>Return type must be same or covariant</em> in method overriding.</td>
</tr>
</tbody>
</table>
12. Explain this Keyword of Java?

Answer:
- It is Used inside any instance method to refer to the current object.
- The value of this refers to the object on which the current method has been called.
- The this keyword can be used where a reference to an object of the current class type is required.
- this keyword : Refer Current Object in Java Programming
- this is keyword in Java.
- We can use this keyword in any method or constructor.
- this keyword used to refer current object.
- Use this keyword from any method or constructor to refer to the current object that calls a method or invokes constructor.

```java
class pixel {
    int x, y;
    void init (int x, int y)
    {
        this.x = x;
        this.y = y;
    }
}

public static void main (String args[])
{
    pixel p = new pixel();
    p.init (4, 3);
}
```

13. What is an exception? Explain the different exception handling mechanisms, with an example, (8M) [Jan 2014, July 2015]

Answer:
- Exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.
- An exception (or exceptional event) is a problem that arises during the execution of a program.
- When an Exception occurs the normal flow of the program is disrupted and the program/Application terminates abnormally, which is not recommended, therefore, these exceptions are to be handled.

- An exception can occur for many different reasons. Following are some scenarios where an exception occurs.
  1. A user has entered an invalid data.
  2. A file that needs to be opened cannot be found.
  3. A network connection has been lost in the middle of communications or the JVM has run out of memory.

**Run time Exception & Compile time Exception**

- Exception Handling is a mechanism to handle runtime errors such as ClassNotFound, IO, SQL, Remote etc.

This is the general form of an exception-handling block:

```
try {
    //block of code to be monitored for errors
}
catch (ExceptionType1 exOb ) {
    //exception handler for ExceptionType1
}
catch (ExceptionType2 exOb ) {
    //exception handler for ExceptionType2
}
//...
finally
{
    //block of code to be executed before try block ends
}
```

**DIVIDE-BY-ZERO EXCEPTION**

This small program has an expression that causes a divide-by-zero error. class DivideByZero

```
public static void main (String args[])
{
    int d = 0;
```
int a = 42 / d;
}
}
The Java run-time system constructs a new exception object when it detects an attempt to divide-by-zero. It then throws this exception. In the above example, there is no exceptional handler to catch the exception. The default handler provided by the Java run-time system will process any exception that is not caught by a program. The output of the above program when executed by the standard Java JDK runtime interpreter:
java.lang.ArithmeticException: / by zero
at DivideByZero.main(DivideByZero.java:4)
Although the default exception handler is provided by Java run-time system, you will usually like to handle exception yourself.
The following program includes a try block and a catch clause, which processes the ArithmeticException generated by the division-by-zero:
```java
class DivideByZero{
    public static void main (String args[]){
        int d, a;
        try //monitor a block of code
        {
            d = 0;
            a = 42 / d;
            System.out.println("This will not be printed. ");
        }
        catch(ArithmeticException e)
        {
            System.out.println("Division by zero.");
        }
        System.out.println("After catch statement.");
    }
    This program generates the following output: Division by zero.
    After catch statement
```
The call to println() inside the try block is never executed. Once an exception is thrown, program control transfers out of the try block into the catch block. Once the catch statement has executed, the program continues with the next line in the program following the entire try/catch mechanism.

14. Explain the types of Exceptions?

Answer:

- There are two types of exceptions
  1. Checked exceptions
  2. Unchecked exceptions

1. Checked/Compiler time exceptions
   - All exceptions other than Runtime Exceptions are known as Checked exceptions
   - The compiler checks exceptions during compilation to see whether the programmer has handled them or not. If these exceptions are not handled/declared in the program, it will give compilation error.
   - Examples of Checked Exceptions: ClassNotFoundException, IllegalAccessException, NoSuchFieldException, EOFException etc

2. Unchecked/Runtime Exceptions
   - Runtime Exceptions are also known as Unchecked Exceptions
   - The compiler do not check whether the programmer has handled them or not but it’s the duty of the programmer to handle these exceptions and provide a safe exit.
   - These exceptions need not be included in any method’s throws list because compiler does not check to see if a method handles or throws these exceptions.
   - Examples of Unchecked Exceptions: ArithmeticException, ArrayIndexOutOfBoundsException, NullPointerException, NegativeArraySizeException etc.
15. Difference between throw and throws in Java

**Answer:**

<table>
<thead>
<tr>
<th>No.</th>
<th><strong>throw</strong></th>
<th><strong>throws</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>1)</td>
<td>Java throw keyword is used to explicitly throw an exception.</td>
<td>Java throws keyword is used to declare an exception.</td>
</tr>
<tr>
<td>2)</td>
<td>Checked exception cannot be propagated using throw.</td>
<td>Checked exception can be propagated with throws.</td>
</tr>
<tr>
<td>3)</td>
<td>Throw is followed by an instance.</td>
<td>Throws is followed by class.</td>
</tr>
<tr>
<td>4)</td>
<td>Throw is used within the method.</td>
<td>Throws is used with the method signature.</td>
</tr>
<tr>
<td>5)</td>
<td>You cannot throw multiple exceptions.</td>
<td>You can declare multiple exceptions e.g. public void method() throws IOException, SQLException</td>
</tr>
</tbody>
</table>

16. Difference between final, finally and finalize

**Answer:**

<table>
<thead>
<tr>
<th>No.</th>
<th><strong>final</strong></th>
<th><strong>finally</strong></th>
<th><strong>finalize</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>1)</td>
<td>Final is used to apply restrictions on class, method and variable. Final class can't be inherited, final method can't be overridden and final variable value can't be changed.</td>
<td>Finally is used to place important code. It will be executed whether exception is handled or not.</td>
<td>Finalize is used to perform clean up processing just before object is garbage collected.</td>
</tr>
<tr>
<td>2)</td>
<td>Final is a keyword.</td>
<td>Finally is a block.</td>
<td>Finalize is a method.</td>
</tr>
</tbody>
</table>
17. **Explain the Abstract class in Java?**

Answer:

- A class that is declared with abstract keyword, is known as abstract class in java. It can have abstract and non-abstract methods (method with body).
- It needs to be extended and its method implemented.

Example abstract class

```java
abstract class A
{
    abstract method
}
```

- A method that is declared as abstract and does not have implementation is known as abstract method.
- Example abstract method

```java
abstract void printStatus();//no body and abstract
```

- Abstraction is a process of hiding the implementation details and showing only functionality to the user.

```java
abstract class Bike
{
    abstract void run();
}
class Honda4 extends Bike
{
    void run()
    {
        System.out.println("running safely..");
    }
    public static void main(String args[])
    {
        Bike obj = new Honda4();
        obj.run();
    }
}
```

18. **Explain the Java interfaces?**

Answer:

- Interface looks like class but it is not a class.
- An interface can have methods and variables just like the class but the methods declared in interface are by default abstract (only method signature, no body).

Also, the variables declared in an interface are public, static & final by default.
NOTE: Compiler automatically converts methods of Interface as public and abstract, and the
data members as public, static and final by default.

Note: class implements interface but an interface extends another interface.

- **Declaration**
  Interfaces are declared by specifying a keyword “interface”. E.g.:

  ```java
  interface MyInterface
  {
  /* All the methods are public abstract by default * 
     Note down that these methods are not 
     having body */
     public void method1();
     public void method2();
  }
  ```

- A class extends another class, an interface extends another interface but a **class implements an interface**.

19. Difference between Abstract class and interfaces?

**Answer:**

<table>
<thead>
<tr>
<th>Abstract class</th>
<th>Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) Abstract class can have <strong>abstract and non-abstract</strong> methods.</td>
<td>Interface can have <strong>only abstract</strong> methods.</td>
</tr>
<tr>
<td>2) Abstract class <strong>doesn't support multiple inheritance</strong>.</td>
<td>Interface <strong>supports multiple inheritance</strong>.</td>
</tr>
<tr>
<td>3) Abstract class <strong>can have final, non-final, static and non-static variables</strong>.</td>
<td>Interface has <strong>only static and final variables</strong>.</td>
</tr>
<tr>
<td>4) Abstract class <strong>can provide the implementation of interface</strong>.</td>
<td>Interface <strong>can't provide the implementation of abstract class</strong>.</td>
</tr>
<tr>
<td>5) The <strong>abstract keyword</strong> is used to declare abstract class.</td>
<td>The <strong>interface keyword</strong> is used to declare interface.</td>
</tr>
</tbody>
</table>
| 6) **Example:**
  ```java
  public abstract class Shape
  {
    public abstract void draw();
  }
  ```
| **Example:**
  ```java
  public interface Drawable
  {
    void draw();
  }
  ```
20. How do you implement the multiple in inheritance in java?

Answer:

- If a class implements multiple interfaces, or an interface extends multiple interfaces i.e. known as multiple inheritance.

```java
interface Moveable
{
    boolean isMoveable();
}

interface Rollable
{
    boolean isRollable
}

class Tyre implements Moveable, Rollable
{
    int width;

    boolean isMoveable()
    {
        return true;
    }

    boolean isRollable()
    {
        return true;
    }

    public static void main(String args[])
    {
        Tyre tr=new Tyre();
        System.out.println(tr.isMoveable());
        System.out.println(tr.isRollable());
    }
}
```
21. Explain the java packages?

Answer:
- Packages in Java is a mechanism to encapsulate a group of classes, interfaces and subpackages.
- A java package is a group of similar types of classes, interfaces and sub-packages.
- A package is a grouping of related types providing access protection and name space management.
- Many implementations of Java use a hierarchical file system to manage source and class files. It is easy to organize class files into packages.
- There are many built-in packages such as java, lang, awt, swing, net, io, util, sql etc.

Advantage of Java Package
1. Java package is used to categorize the classes and interfaces so that they can be easily maintained.
2. Java package provides access protection.
3. Java package removes naming collision
4. Reusability of code is one of the most important requirements in the software industry. Reusability saves time, effort and also ensures consistency. A class once developed can be reused by any number of programs wishing to incorporate the class in that particular program.

Types of Packages
1. Built-in Packages
   - These packages consist of a large number of classes which are a part of Java API.
   - Some of the commonly used built-in packages are:
     1) java.lang: Contains language support classes (e.g., classes which define primitive data types, math operations). This package is automatically imported.
     2) java.io: Contains classes for supporting input / output operations.
     3) java.util: Contains utility classes which implement data structures like Linked List, Dictionary and support; for Date / Time operations.
     4) java.applet: Contains classes for creating Applets.
     5) java.awt: Contain classes for implementing the components for graphical user interfaces (like button , menus etc).
     6) java.net: Contain classes for supporting networking operations.

2. User-defined packages
   - These are the packages that are defined by the user.

Simple example of java package
- The package keyword is used to create a package in java.
  ```java
  //save as Simple.java
  package mypack;
  public class Simple
  {
      public static void main(String args[])
      {
          System.out.println("Welcome to package");
      }
  }
  ```
22. How to import the package from another package?

Answer:

There are three ways to access the package from outside the package.

I. import package.*;

II. import package.classname;

III. fully qualified name.

Using packagename.classname

- If you import package.classname then only declared class of this package will be accessible.

Example of package by import package.classname

//save by A.java
package pack;
public class A
{
    public void msg()
    {
        System.out.println("Hello");
    }
}

//save by B.java
package mypack;
import pack.A;
class B
{
    public static void main(String args[])
    {
        A obj = new A();
        obj.msg();
    }
}

Using fully qualified name

- If we use fully qualified name then only declared class of this package will be accessible. Now there is no need to import.
  - But you need to use fully qualified name every time when you are accessing the class or interface.
  - It is generally used when two packages have same class name
    e.g. java.util and java.sql packages contain Date class.

Example of package by import fully qualified name

//save by A1.java
package pack;
public class A1
{
    public void msg(){System.out.println("Hello");}
}

//save by B1.java
package mypack;
class B1
{
public static void main(String args[]){
    obj.msg();
}  }

23. Explain the package Access Specifiers?

   Answer:
   I. private: accessible only in the class
   II. no modifier: so-called “package” access — accessible only in the same package
   III. protected: accessible (inherited) by subclasses, and accessible by code in same package
   IV. public: accessible anywhere the class is accessible, and inherited by subclasses

<table>
<thead>
<tr>
<th>Location</th>
<th>private</th>
<th>No modifier</th>
<th>protected</th>
<th>public</th>
</tr>
</thead>
<tbody>
<tr>
<td>Same class</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Same package and also a subclass</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Same package but not a subclass</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Different package but a subclass</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Different package but not a subclass</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>
MODLE 4

MULTI THREADED PROGRAMMING, EVENT HANDLING

1. What is synchronization? Explain with an example, how synchronization is implemented in Java. (6M) [Jan 2014, July 2016, Jan 2015]

Answer:
One further area of concern within threads is known as "busy waiting." This is the situation where a thread is conceptually idle, perhaps waiting for some other synchronous processing to complete, but yet it is still occupying the CPU. To illustrate, consider a spell-checking thread. It reads a text file and searches for each word in its database. If a word is not found, the thread composes a list of suggested corrections and notifies the calling process. The calling process displays a list of the while the thread waits. Eventually the user makes a selection and the calling process allows the thread to continue. The thread eventually terminates, once the whole file has been processed.

We might write such code by using a public boolean variable, 'paused,' like so: paused = true;
parent.processWordList (aListOfWords);
// loop until parent process clears 'paused'
while (paused && !terminated) {} // Continue
The thread simply loops continually until the paused variable is set to false. Although this seems intuitive, the ramifications are that the CPU will continue to spend significant time processing the looping time that could best be spent servicing other threads. Indeed, any attempt to move windows around on the screen will be noticeably jerky.

void sleep (long milliseconds)
void sleep (long milliseconds, int nanoseconds)
These methods make the thread pause for the specified number of milliseconds and/or nanoseconds. What is important here is that the thread really does pause and takes no CPU time. Not all operating systems support time resolutions as small as nanoseconds, and in
these cases the second method simply rounds the number of nanoseconds to the nearest millisecond and calls the first method.

The sleep methods have been defined in the Java class libraries as being able to throw an `InterruptedException`. This exception is generated if the sleep method is disturbed in some way before it completes, such as if `System.exit()` is called from elsewhere in the application, shutting the program down. Whether or not you wish to perform any processing to respond specifically to a sleep being interrupted, Java mandates the exception be caught, hence your calls to the sleep methods should be wrapped in a `try/catch` block. This exception was designed to provide a general mechanism to allow one thread to implement another. Unfortunately this has not yet been fully implemented.

The thread class provides a method, `interrupt()`, which sends an interruption to a specified thread presently this amounts to nothing more than setting a boolean flag. The boolean function `isInterrupted()` may be used to query the status of this flag, but unfortunately there is not presently any way to actually interrupt a thread and throw the `InterruptedException`. So despite the fact that the exception must be caught it currently isn't useful for anything. Eventually, it will permit threads to be woken up from their sleep. Because the sleep method is not presently interruptible, the alternative is to have brief periods of inactivity (sleeping), before querying the paused status. Our code fragment thus becomes:

```java
paused = true;
parent.processWordList (aListOfWords);
// loop until parent process clears 'paused' while (paused && !terminated)
try { sleep (2000); }
catch (InterruptedException e) {} // Continue
```

This code tells the thread to sleep for two seconds (2000 milliseconds) in the body of the while loop. The thread continues to loop but puts much less strain on the CPU. In fact, the thread's awake time is greatly reduced.

You have to be careful, though, not to make the sleep time too long or the thread will not respond swiftly once the pause flag has been cleared. Because you are using the thread's sleep method, you have to catch the exception that could be raised (or javac will complain), but you don't need to specify any code in the body of the exception handler.

Answer:

Problem Description:

In computer science the producer-consumer problem (also known as the bounded-buffer problem) is a classical example of a multi-process synchronization problem. The problem describes two processes, the producer and the consumer, who share a common, fixed-size buffer.

The producer's job is to generate a piece of data, put it into the buffer and start again. At the same time the consumer is consuming the data (i.e. removing it from the buffer) one piece at a time. The problem is to make sure that the producer won't try to add data into the buffer if it's full and that the consumer won't try to remove data from an empty buffer. This is the code for solving the above stated:

```java
class BufferItem {
    public volatile double value = 0; // multiple threads access
    public volatile boolean occupied = false; // so make these `volatile'
}

class BoundedBuffer { // designed for a single producer thread and // a single consumer thread
    private int numSlots = 0;
    private BufferItem[] buffer = null;
    private int putIn = 0, takeOut = 0;
    // private int count = 0;
    public BoundedBuffer(int numSlots) {
        if (numSlots <= 0) throw new IllegalArgumentException("numSlots<=0");
        this.numSlots = numSlots;
        buffer = new BufferItem[numSlots];
        for (int i = 0; i < numSlots; i++) buffer[i] = new BufferItem();
        // count++; // race condition!!!
    }
    public double fetch() {
        double value;
        while (!buffer[takeOut].occupied) // busy wait
            Thread.currentThread().yield();
        value = buffer[takeOut].value; // C
    }
}
```

In computer science the producer-consumer problem (also known as the bounded-buffer problem) is a classical example of a multi-process synchronization problem. The problem describes two processes, the producer and the consumer, who share a common, fixed-size buffer.
buffer[takeOut].occupied = false; // D takeOut = (takeOut + 1) % numSlots;
// count--; // race condition!!! return value;
}
}

3. **What is delegation event model? Describe the significance of adapter class, with an example. (6M) [Jan 2014, Jan 2016, July 2016]**

**Answer:**
In Java, events represent all activity that goes on between the user and the application.

**Two event handling mechanisms:**
Delegation event model: It defines standard and consistent mechanisms to generate and process events. Here the source generates an event and sends it to one or more listeners. The listener simply waits until it receives an event. Once it is obtained, it processes this event and returns. Listeners should register themselves with a source in order to receive an event notification. Notifications are sent only to listeners that want to receive them.

**Events**
In the delegation model, an *event* is an object that describes a state change in a source. It can be generated as a consequence of a person interacting with the elements in a graphical user interface. Some of the activities that cause events to be generated are: pressing a button, entering a character via the keyboard, selecting an item in a list, and clicking the mouse. Events may also occur that are not directly caused by interactions with a user interface. For example, an event may be generated when a timer expires, a counter exceeds a value, a software or hardware failure occurs, or an operation is completed.

**Event Classes**
The classes that represent events are at the core of Java’s event handling mechanism.

**EventObject:** It is at the root of the Java event class hierarchy in `java.util`. It is the superclass for all events. Its one constructor is shown here: `EventObject(Object src)`
Here, `src` is the object that generates this event. `EventObject` contains two methods: `getSource()` and `toString()`. The `getSource()` method returns the source of the event.

**EventObject** is a superclass of all events.

**The ActionEvent Class:**
An *ActionEvent* is generated when a button is pressed, a list item is double-clicked, or a menu item is selected. The *ActionEvent* class defines four integer constants that can be
used to identify any modifiers associated with an action event: ALT_MASK, CTRL_MASK, META_MASK, and SHIFT_MASK.

**ActionEvent** has these three constructors: ActionEvent(Object src, int type, String cmd) ActionEvent(Object src, int type, String cmd, int modifiers)

ActionEvent(Object src, int type, String cmd, long when, int modifiers) Here, src is a reference to the object that generated this event. The type of the event is specified by type, and its command string is cmd. The argument modifiers indicates which modifier keys (ALT, CTRL, META, and/or SHIFT) were pressed when the event was generated.

The when parameter specifies when the event occurred

- **The AdjustmentEvent Class** An AdjustmentEvent is generated by a scroll bar
- **The ComponentEvent Class** A ComponentEvent is generated when the size, position, or visibility of a component is changed. There are four types of component events
  - The ContainerEvent Class A ContainerEvent is generated when a component is added to or removed from a container
  - The FocusEvent Class : A FocusEvent is generated when a component gains or loses input focus
  - The InputEvent Class : The abstract class InputEvent is a subclass of ComponentEvent and is the superclass for component input events. Its subclasses are KeyEvent and MouseEvent.
  - The ItemEvent Class : An ItemEvent is generated when a check box or a list item is clicked or when a checkable menu item is selected or deselected
  - The KeyEvent Class A KeyEvent is generated when keyboard input occurs.
  - The MouseEvent Class There are eight types of mouse events
    - The MouseWheelEvent Class The MouseWheelEvent class encapsulates a mouse wheel event.
  - The TextEvent Class Instances of this class describe text events. These are generated by text fields and text areas when characters are entered by a user or program.
  - **The WindowEvent Class** There are ten types of window events. The WindowEvent class defines integer constants that can be used to identify them.
4. What is meant by multithreaded programming? Explain with an example interthread communication [July 2014, Jan 2014, Jan 2016]

Answer:
A thread is a single path of execution of code in a program.

- A Multithreaded program contains two or more parts that can run concurrently.
- Each part of such a program is called a Thread.
- Each thread defines a separate path of execution. Multithreading is a specialized form of Multitasking.

to make the classes threadable
A class can be made threadable in one of the following ways
(1) implement the Runnable Interface and apply its run() method.
(2) extend the Thread class itself.

1. Implementing Runnable Interface: The easiest way to create a thread is to create a class that implements the Runnable interface. To implement Runnable, a class need only implement a single method called run().
The Format of that function is public void run().

2. Extending Thread: The second way to create a thread is to create a new class that extends the Thread class and then to create an instance of this class. This class must override the run() method which is the entry point for the new thread.

5. What is meant by thread priority? How it is assigned [July 2014 Jan 2015]

Answer:
The example in the next segment demonstrates the use of Runnable and its implementation.

Synchronization
Two or more threads accessing the same data simultaneously may lead to loss of data integrity. In order to avoid this java uses the concept of monitor. A monitor is an object used as a mutually exclusive lock.
At a time only one thread can access the Monitor. A second thread cannot enter the monitor until the first comes out. Till such time the other thread is said to be waiting.
The keyword Synchronized is used in the code to enable synchronization and it can be used along with a method.

**Changing the state of thread**

There might be times when you need to temporarily stop a thread from processing and then resume processing, such as when you want to let another thread use the current resource. You can achieve this objective by defining your own suspend and resume methods, as shown in the following example. This example defines a MyThread class. The MyThread class defines three methods: the run() method, the suspendThread() method, and the resumeThread() method. In addition, the MyThread class declares the instance variable suspended, whose value is used to indicate whether or not the thread is suspended.

class MyThread implements Runnable
{
    String name;
    Thread t;
    boolean suspended;
    MyThread()
    {
        t = new Thread(this, "Thread");
        suspended = false; t.start();
    }
    public void run()
    { try {
        for (int i = 0; i < 10; i++) { System.out.println("Thread: "+i); Thread.sleep(200);
            synchronized (this) {
                while (suspended) {
                    wait();
                }
            }
        }
    } catch (InterruptedException e) { System.out.println("Thread: interrupted.");
        System.out.println("Thread exiting.");
    }
    void suspendThread() { suspended = true; }
}
synchronized void resumeThread() {
    suspended = false;
    notify();
}

class Demo {
    public static void main(String args[]) {
        MyThread t1 = new MyThread();
        try {
            Thread.sleep(1000); t1.suspendThread(); System.out.println("Thread: Suspended");
            Thread.sleep(1000);
            t1.resumeThread(); System.out.println("Thread: Resume");
        } catch (InterruptedException e) {
        }
        try {
            t1.t.join();
        } catch (InterruptedException e) {
            System.out.println("Main Thread: interrupted");
        }
    }
}

6. Explain action event class & adjustment event class (4M) [July 2014 Jan 2015]

Answer:

EventListener Interfaces

Listeners are created by implementing one or more of the interfaces defined by the java.awt.event package.

When an event occurs, the event source invokes the appropriate method defined by the listener and provides an event object as its argument.

Interface Description

**ActionListener** - Defines one method to receive action events.

**AdjustmentListener** - Defines one method to receive adjustment events.

**ComponentListener** - Defines four methods to recognize when a component is hidden, moved, resized, or shown.
**ContainerListener** - Defines two methods to recognize when a component is added to or removed from a container.

**FocusListener** - Defines two methods to recognize when a component gains or loses keyboard focus.

**ItemListener** - Defines one method to recognize when the state of an item changes.

**KeyListener** - Defines three methods to recognize when a key is pressed, released, or typed.

**MouseListener** - Defines five methods to recognize when the mouse is clicked, enters a component, exits a component, is pressed, or is released.

**MouseMotionListener** - Defines two methods to recognize when the mouse is dragged or moved.

**MouseListener** - Defines one method to recognize when the mouse wheel is moved.

**TextListener** - Defines one method to recognize when a text value changes.

**WindowFocusListener** - Defines two methods to recognize when a window gains or loses input focus.

**WindowListener** Defines seven methods to recognize when a window is activated, closed, deactivated, deiconified, iconified, opened

7. **Short notes on static, Final, Abstract, Native (10M) [July 2015, Jan 2016]**

**Answer:**

A nested class has the same behavior as any static member of a class. You can have access to it without initializing the *parent* class, and they can access the *parent* class static methods and variables also. Nested classes are always define with the keyword **static** (and we will see later that this is what differentiate them from the inner classes). An access tag (i.e. : public, protected or private) can be defined, but by default a nested class takes the default package access. Sun considers nested classes to be top-level classes (I find this at times to be confusing). Here is an example of how to define a nested class MyInner in the class enclosing class MyOuter.

Notice that when you compile this code, you will have two .class file as the output :

MyOuter.class : being the enclosing class.

MyOuter$MyInner.class : being the inner class. Since class MyInner is a static member of MyOuter, it can be access from anywhere in your code using MyOuter.MyInner (The same
way that you access classes in packages, you can even use the import statement with nested classes: import MyOuter.MyInner).

8. **What is a thread? explain 2 ways of creating thread (10M) [July 2014, July 2015]**

**Answer:**
A thread is a single path of execution of code in a program.

- A Multithreaded program contains two or more parts that can run concurrently.
- Each part of such a program is called a Thread.
- Each thread defines a separate path of execution. Multithreading is a specialized form of Multitasking.

**to make the classes threadable**
A class can be made threadable in one of the following ways

1. Implement the Runnable Interface and apply its run() method.
2. Extend the Thread class itself.

3. Implementing Runnable Interface: The easiest way to create a thread is to create a class that implements the Runnable interface. To implement Runnable, a class need only implement a single method called run().
   The Format of that function is `public void run()`.

4. Extending Thread: The second way to create a thread is to create a new class that extends the Thread class and then to create an instance of this class. This class must override the run() method which is the entry point for the new thread.
MODULE 5

THE APPLET CLASS:

1. Write an applet program to display the message “VTU BELGAUM”. Set the background color to cyan and foreground color to red. (6M) [Jan 2014, Jan 2016, July 2016]

Answer:
The applet parameter "Message" is the string to be drawn.
import java.applet. *; import java.awt. *
public class DrawStringApplet extends Applet {
    private String defaultMessage = "Hello!";
    public void paint(Graphics g) {
        String inputFrom = this.getParameter("Message");
        if (inputFrom == null) inputFrom = defaultMessage;
        g.drawString(inputFrom, 50, 25);
    }
}

HTML file that references the above applet.
<HTML> <HEAD>
<TITLE> Draw String </TITLE>
</HEAD>
<BODY>
This is the applet:<P>
<APPLET code="DrawStringApplet" width="300" height="50">
    <PARAM name="Message" value="Howdy, there!"> This will be very boring if your browser doesn't understand Java. </APPLET>
</BODY> </HTML>
2. What is an applet? Different stages of an applet(8M) [July 2014, Jan 2015 Jan 2016]

Answer:
· Applet is a special type of JAVA program i.e embedded in the webpage to generate the dynamic content.
· It runs inside the browser and works at client side.
· All applets are subclasses (either directly or indirectly) of Applet.
· Applets are not stand-alone programs. Instead, they run within either a web browser or an applet viewer.
· An applet is a Java program that runs in a Web browser.
· An applet can be a fully functional Java application because it has the entire Java API.
· Execution of an applet does not begin at main( ). Actually, few applets even have main( ) methods. Instead, execution of an applet is started and controlled with an entirely different mechanism.
· Output to your applet's window is not performed by System.out.println( ). Rather, in non-Swing applets, output is handled with various AWT methods, such as drawString( ), which outputs a string to a specified X,Y location.
· Input is also handled differently than in a console application.
· To use an applet, it is specified in an HTML file. One way to do this is by using the APPLET tag.
· The applet will be executed by a Java-enabled web browser when it encounters the APPLET tag within the HTML file

Different Stages of an Applet (Lifecycle of Java Applet)
1. Applet is initialized.
2. Applet is started.
3. Applet is painted.
4. Applet is stopped.
5. Applet is destroyed.

1. public void init(): is used to initialized the Applet. It is invoked only once.
2. public void start(): is invoked after the init() method or browser is maximized. It is used to start the Applet.
3. public void stop(): is used to stop the Applet. It is invoked when Applet is stop or browser is minimized.

4. public void destroy(): is used to destroy the Applet. It is invoked only once.

java.awt (Component class)

5. public void paint(Graphics g):
   It is used to paint the Applet. It provides Graphics class object that can be used for drawing oval, rectangle, arc etc.

Example of Simple Applet

```java
import java.awt.*;
import java.applet.**;
public class FirstApplet extends Applet
{
    String str;
    public void init()
    {
        str = "Java is interesting!";
    }
    public void paint(Graphics g)
    {
        g.drawString(str, 70, 80);
    }
}
```
3. **Explain the Types of Applets and HTML applet tag?**

**Answer:**

Two varieties of applets.
- The **first** are those based directly on the Applet class.
  - The applets use the **Abstract Window Toolkit (AWT)** to provide the graphic user interface (or use no GUI at all).
  - This style of applet has been available since Java was first created.
- The **second type of applets** are those based on the **Swing class JApplet**.
  - Swing applets use the Swing classes to provide the GUI.
  - Swing offers a richer and often easier-to-use user interface than does the AWT.
  - Thus, Swing-based applets are now the most popular.

- The APPLET tag be used to start an applet from both an HTML document and from an applet viewer.
- An applet viewer will execute each APPLET tag that it finds in a separate window, while web browsers will allow many applets on a single page.
- The syntax for a fuller form of the APPLET tag is shown here.

```
<APPLET
  [CODEBASE = codebaseURL]
  CODE = appletFile
  [ALT = alternateText]
  [NAME = appletInstanceName]
  WIDTH = pixels HEIGHT = pixels
  [ALIGN = alignment]
  [VSPACE = pixels] [HSPACE = pixels]
>
[< PARAM NAME = AttributeName VALUE = AttributeValue>]
[< PARAM NAME = AttributeName2 VALUE = AttributeValue>]
...
[HTML Displayed in the absence of Java]
</APPLET>
```

Where

- **CODEBASE**
  - CODEBASE is an optional attribute
  - Specifies the base URL of the applet code.
  - The HTML document's URL directory is used as the CODEBASE if this attribute is not specified. The CODEBASE does not have to be on the host from which the HTML document was read.

- **CODE**
  - CODE is a required attribute
  - It gives the name of the file containing your applet's compiled .class file.
  - This file is relative to the code base URL of the applet, which is the directory that the HTML file was in or the directory indicated by CODEBASE if set.

- **ALT**
  - The ALT tag is an optional attribute
  - It is used to specify a short text message that should be displayed if the browser recognizes the APPLET tag
  - It can't currently run Java applets.
This is distinct from the alternate HTML you provide for browsers that don't support applets.

- **NAME**
  - NAME is an optional attribute
  - It is used to specify a name for the applet instance.
  - Applets must be named in order for other applets on the same page to find them by name and communicate with them.
  - To obtain an applet by name, use `getApplet()` method, which is defined by the `AppletContext` interface.

- **WIDTH and HEIGHT**
  - WIDTH and HEIGHT are required attributes that give the size (in pixels) of the applet display area.

- **ALIGN**
  - ALIGN is an optional attribute
  - It specifies the alignment of the applet.
  - This attribute is treated the same as the HTML IMG tag with these possible values:
    - `LEFT`, `RIGHT`, `TOP`, `BOTTOM`, `MIDDLE`, `BASELINE`, `TEXHTOP`, `ABSMIDDLE`, and `ABSBOTTOM`.

- **VSPACE and HSPACE**
  - These attributes are optional.
  - VSPACE specifies the space, in pixels, above and below the applet.
  - HSPACE specifies the space, in pixels, on each side of the applet.
  - They're treated the same as the IMG tag's VSPACE and HSPACE attributes.

- **PARAM NAME and VALUE**
  - The PARAM tag allows you to specify applet-specific arguments in an HTML page.
  - Applets access their attributes with the `getParameter()` method.
4. Write the difference between stand alone application and applet?

Answer:

<table>
<thead>
<tr>
<th>FEATURE</th>
<th>STAND ALONE APPLICATION</th>
<th>APPLET</th>
</tr>
</thead>
<tbody>
<tr>
<td>main() method</td>
<td>Present</td>
<td>Not present</td>
</tr>
<tr>
<td>Execution</td>
<td>Requires JRE</td>
<td>Requires a browser like Chrome</td>
</tr>
<tr>
<td>Nature</td>
<td>Called as stand-alone application as application can be executed from command prompt</td>
<td>Requires some third party tool help like a browser to execute</td>
</tr>
<tr>
<td>Restrictions</td>
<td>Can access any data or software available on the system</td>
<td>cannot access anything on the system except browser’s services</td>
</tr>
<tr>
<td>Security</td>
<td>Does not require any security</td>
<td>Requires highest security for the system as they are untrusted</td>
</tr>
</tbody>
</table>

5. How do you Pass the Parameters to Applets?

Answer:

- The APPLET tag in HTML allows you to pass parameters to your applet.
- To retrieve a parameter, use the getParameter() method.
- It returns the value of the specified parameter in the form of a String object.
- Syntax: public String getParameter(String parameterName)

import java.applet.Applet;
import java.awt.Graphics;
public class UseParam extends Applet
{
    public void paint(Graphics g)
    {
        String str=getParameter("msg");
        g.drawString(str,50,50);
    }
}
myapplet.html
<html>
<body>
<applet code="UseParam.class" width="300" height="300">
<param name="msg" value="Welcome to applet">
</applet>
</body>
</html>
6. Write Short notes on . getDocumentBase( ) and getCodeBase( )?

Answer:
- We will create applets that will need to explicitly load media and text.
- Java will allow the applet to load data from
  The directory holding the HTML file that started the applet (the document base) and
- The directory from which the applet's class file was loaded (the code base).
- These directories are returned as URL objects by getDocumentBase( ) and getCodeBase( ).
- They can be concatenated with a string that names the file you want to load.

```java
// Display code and document bases.
import java.awt.*;
import java.applet.*;
import java.net.*;
/**
<applet code="Bases" width=300 height=50>
</applet>
*/
public class Bases extends Applet
{
    // Display code and document bases.
    public void paint(Graphics g)
    {
        String msg;
        URL url = getCodeBase(); // get code base
        msg = "Code base: " + url.toString();
        g.drawString(msg, 10, 20);
        url = getDocumentBase(); // get document base
        msg = "Document base: " + url.toString();
        g.drawString(msg, 10, 40);
    }
}
```

7. Write short noted on . AppletContext and showDocument( )?

Answer:
- Java allows the applet to transfer the control to another URL by using the showDocument( ) Method defined in the AppletContext interface.
- For this, first of all, it is needed to obtain the Context of the currently executing applet by calling the getAppletContext() method defined by the Applet.
- Once the context of the applet is obtained with in an applet, another document can be brought into view by calling showDocument() method.
- There are two showDocument() methods which are as follows:
showDocument(URL url)
showDocument(URL url,string loc)

where,
url is the URL from where the document is to be brought into view.
loc is the location within the browser window where the specified document is to be displayed.

Using an applet context, getCodeBase(), and showDocument() to display an HTML file.*/
import java.awt.*/;
import java.applet.*/;
import java.net.*/;
/*
  <applet code="ACDemo" width=300 height=50>
  </applet>
*/
public class ACDemo extends Applet
{
  public void start()
  {
    AppletContext ac = getAppletContext();
    URL url = getCodeBase(); // get url of this applet
    try {
      ac.showDocument(new URL(url+"Test.html"));
    }
    catch(MalformedURLException e)
    {
      showStatus("URL not found");
    }
  }
}
SWINGS

1. What is swing? List the main swing features. Explain the different types of panes of swing containers. (10M) [Jan 2014, July 2016, July 2015]

Answer:

- Swing is a set of classes that provides more powerful and flexible GUI components than does the AWT.
- Swing provides the look and feel of the modern Java GUI.
- The number of classes and interfaces in the Swing packages is quite large.

Two Key Swing Features

- Swing was created to address the limitations present in the AWT. It does this through two key features:
  - **Lightweight components**
  - **A pluggable look and feel.**

1. **Swing Components Are Lightweight**

- Swing components are *lightweight*. This means that they are written entirely in Java and do not map directly to platform-specific peers.
- Lightweight components are rendered using graphics primitives, they can be transparent, which enables nonrectangular shapes.
- Lightweight components are more efficient and more flexible.
- Lightweight components do not translate into native peers, the look and feel of each component is determined by Swing, not by the underlying operating system. This means that each component will work in a consistent manner across all platforms.

2. **Swing Supports a Pluggable Look and Feel(PLAF)**

- Swing supports a *pluggable look and feel* (PLAF).
- Each Swing component is rendered by Java code rather than by native peers, the look and feel of a component is under the control of Swing.
- To separate the look and feel of a component from the logic of the component, and this is what Swing does.
- A component is rendered without affecting any of its other aspects.
- In other words, it is possible to "plug in" a new look and feel for any given component without creating any side effects in the code that uses that component.
- Moreover, it becomes possible to define entire sets of look-and-feels that represent different GUI styles.
To use a specific style, its look and feel is simply "plugged in." Once this is done, all components are automatically rendered using that style.

Types of panes

- Swing is built on top of AWT and is entirely written in Java, using AWT’s lightweight component support.
- In particular, unlike AWT, the architecture of Swing components makes it easy to customize both their appearance and behavior.
- Components from AWT and Swing can be mixed, allowing you to add Swing support to existing AWT-based programs.
- For example, swing components such as JSlider, JButton and JCheckbox could be used in the same program with standard AWT labels, textfields and scrollbars.

Component set (subclasses of JComponent) Support classes, Interfaces

Swing Components and Containers

Swing components are basic building blocks of an application. Swing toolkit has a wide range of various widgets. Buttons, check boxes, sliders, list boxes etc. Everything a programmer needs for his job. In this section of the tutorial, we will describe several useful components.

JLabel Component

JLabel is a simple component for displaying text, images or both. It does not react to input events.

JCheckBox

JCheckBox is a widget that has two states. On and Off. It is a box with a label. JSlider

JSlider is a component that lets the user graphically select a value by sliding a knob within a bounded interval.

 JComboBox

Combobox is a component that combines a button or editable field and a drop-down list. The user can select a value from the drop-down list, which appears at the user’s request.

JProgressBar

A progress bar is a widget that is used, when we process lengthy tasks. It is animated so that the user knows, that our task is progressing.

JToggleButton
2. Create a swing application having two buttons named alpha and beta. When either of the buttons pressed, it should display “alpha pressed” and “beta pressed” respectively. (10M) [Jan 2014, July 2015, Jan 2016]

Answer:

Create a JLabel with an image

```java
import java.awt.FlowLayout;
import java.awt.HeadlessException;
import javax.swing.Icon;
import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JLabel;

public class Main extends JFrame {
    public Main() throws HeadlessException {
        setSize(300, 300);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new FlowLayout(FlowLayout.LEFT));
        Icon icon = new ImageIcon("a.png");
        JLabel label1 = new JLabel("Full Name :", icon, JLabel.LEFT);
        JLabel label2 = new JLabel("Address :", JLabel.LEFT);
        label2.setIcon(new ImageIcon("b.png"));
        getContentPane().add(label1);
        getContentPane().add(label2);
    }

    public static void main(String[] args) {
        new Main().setVisible(true);
    }
}
```
3. List the different types of swing buttons. Write a program to create four types of buttons on JApplet. Use suitable events to show actions on the buttons and use JLabel to display the action invoked. (10M) [Jan 2014, July 2016]

Answer:

```java
package com.ack.gui.swing.simple;

import java.awt.*;
import java.awt.event.WindowAdapter; import java.awt.event.WindowEvent; import javax.swing.*;

public class SimpleSwingButtons extends JFrame {
    public static void main( String[] argv ) {
        SimpleSwingButtons myExample = new SimpleSwingButtons( "Simple Swing Buttons" );
    }
    public SimpleSwingButtons( String title ) {
        super( title );
        setSize( 150, 150 );
        addWindowListener( new WindowAdapter() { public void windowClosing( WindowEvent we ) { dispose();
            System.exit( 0 );
        } });
        init();
        setVisible( true );
    }
    private void init() {
        JPanel my_panel = new JPanel();
        my_panel.setLayout( new GridLayout( 3, 3 ) ); for( int i = 1; i < 10; i++ ) {
            ImageIcon icon = new ImageIcon( i + "gif" ); JButton jb = new JButton( icon ); jb.setToolTipText( i + "gif" );
            my_panel.add( jb );
        }
        getContentPane().add( my_panel );
    }
}
```
my_panel.setBorder( BorderFactory.createEtchedBorder() );
}
]

4. Write the steps to create Jtable. WAP to create a table with the column headings Name, USN, age, address & insert records and display(10M) [July 2014, Jan 2015]

Answer:
The JTable is used to display and edit regular two-dimensional tables of cells. The JTable has many facilities that make it possible to customize its rendering and editing but provides defaults for these features so that simple tables can be set up easily. For example, to set up a table with 10 rows and 10 columns of numbers:

```
TableModel dataModel = new AbstractTableModel() {
    public int getColumnCount() { return 10;
}  public int getRowCount() { return 10; }
    public Object getValueAt(int row, int col) { return new Integer(row*col); }
};
JTable table = new JTable(dataModel);
JScrollPane scrollpane = new JScrollPane(table);
```

Note that if you wish to use a JTable in a standalone view (outside of a JScrollPane) and want the header displayed, you can get it using `getTableHeader()` and display it separately.

To enable sorting and filtering of rows, use a RowSorter. You can set up a row sorter in either of two ways:
- Directly set the RowSorter. For example: `table.setRowSorter(new TableRowSorter(model));`
- Set the `autoCreateRowSorter` property to true, so that the JTable creates a RowSorter for you. For example: `setAutoCreateRowSorter(true);`

When designing applications that use the JTable it is worth paying close attention to the data structures that will represent the table’s data. The DefaultTableModel is a model implementation that uses a Vector of Vectors of Objects to store the cell values. As well as copying the data from an application into the DefaultTableModel, it is also possible to wrap the data in the methods of the TableModel interface so that the data can be passed to the JTable directly, as in the example above. This often results in more efficient applications because the model is free to choose the internal representation that best suits the data. A good
rule of thumb for deciding whether to use the AbstractTableModel or the DefaultTableModel is to use the AbstractTableModel as the base class for creating subclasses and the DefaultTableModel when subclassing is not required.

The "TableExample" directory in the demo area of the source distribution gives a number of complete examples of JTable usage, covering how the JTable can be used to provide an editable view of data taken from a database and how to modify the columns in the display to use specialized renderers and editors.

The JTable uses integers exclusively to refer to both the rows and the columns of the model that it displays. The JTable simply takes a tabular range of cells and uses getValueAt(int, int) to retrieve the values from the model during painting. It is important to remember that the column and row indexes returned by various JTable methods are in terms of the JTable (the view) and are not necessarily the same indexes used by the model.

By default, columns may be rearranged in the JTable so that the view's columns appear in a different order to the columns in the model. This does not affect the implementation of the model at all: when the columns are reordered, the JTable maintains the new order of the columns internally and converts its column indices before querying the model.

So, when writing a TableModel, it is not necessary to listen for column reordering events as the model will be queried in its own coordinate system regardless of what is happening in the view. In the examples area there is a demonstration of a sorting algorithm making use of exactly this technique to interpose yet another coordinate system where the order of the rows is changed, rather than the order of the columns.

5. Difference between swings and AWT (10M) [Jan 2015, July 2014, Jan 2016]

Answer:
public class JTextField
extends JTextComponent
implements SwingConstants

JTextField is a lightweight component that allows the editing of a single line of text. For information on and examples of using text fields, JTextField is intended to be source-compatible with java.awt.TextField where it is reasonable to do so. This component has capabilities not found in the java.awt.TextField class. The superclass should be consulted for additional capabilities.
JTextField has a method to establish the string used as the command string for the action event that gets fired. The java.awt.TextField used the text of the field as the command string for the ActionEvent. JTextField will use the command string set with the setActionCommand method if not null, otherwise it will use the text of the field as a compatibility with java.awt.TextField.

The method setEchoChar and getEchoChar are not provided directly to avoid a new implementation of a pluggable look-and-feel inadvertently exposing password characters. To provide password-like services a separate class JPasswordField extends JTextField to provide this service with an independently pluggable look-and-feel.

The java.awt.TextField could be monitored for changes by adding a TextListener for TextEvent's. In the JTextComponent based components, changes are broadcasted from the model via a DocumentEvent to DocumentListeners. The DocumentEvent gives the location of the change and the kind of change if desired. The code fragment might look something like:

```
DocumentListener myListener = ??;
JTextField myArea = ??;
myArea.getDocument().addDocumentListener(myListener);
```

The horizontal alignment of JTextField can be set to be left justified, leading justified, centered, right justified or trailing justified. Right/trailing justification is useful if the required size of the field text is smaller than the size allocated to it. This is determined by the setHorizontalAlignment and getHorizontalAlignment methods. The default is to be leading justified.

How the text field consumes VK_ENTER events depends on whether the text field has any action listeners. If so, then VK_ENTER results in the listeners getting an ActionEvent, and the VK_ENTER event is consumed. This is compatible with how AWT text fields handle VK_ENTER events. If the text field has no action listeners, then as of v 1.3 the VK_ENTER event is not consumed. Instead, the bindings of ancestor components are processed, which enables the default button feature of JFC/Swing to work.

Customized fields can easily be created by extending the model and changing the default model provided. For example, the following piece of code will create a field that holds only upper case characters. It will work even if text is pasted into from the clipboard or it is altered via programmatic changes.