1)A super class

B extend A

C extend A

D extend C

E extend C

2)

3)Inheritance helps the code to be reused in many situations.the base calss is defines and once it is compiled.it need not be reworked .using the concept of inheritance ,the programmer can create as many derived classes from the base class as needed while adding specific features to each derived class as needed.

The above Concept of reusability achieved by inheritance saves the progrramers time and effort . Since the main code written can be reused in various situations as needed.

5)

**public** **class** Java

{

}

**class** Shape

{

}

**class** TwoDShape **extends** Shape

{

}

**class** circle **extends** TwoDShape

{

}

**class** scqure **extends** TwoDShape

{

}

**class** trangle **extends** TwoDShape

{

}

**class** ThreeDShape **extends** Shape

{

}

**class** sphere **extends** ThreeDShape

{

}

**class** cube **extends** ThreeDShape

{

}

**class** tetraheaden **extends** ThreeDShape

{

}

6)

**public** **class** Java

{

**public** **static** **void** main(String args[])

{

}

}

**class** Student

{

**int** s;

**public** **void** printS()

{

System.***out***.println("Students"+s);

}

}

**class** UndergraduateStudent **extends** Student

{

**int** u;

**public** **void** printUs()

{

System.***out***.println("UndergraduateStudent"+s);

}

}

**class** Freshman **extends** UndergraduateStudent

{

**int** u;

**public** **void** printF()

{

System.***out***.println("UndergraduateStudent"+u);

}

}

**class** Sophomore **extends** UndergraduateStudent

{

**int** so;

**public** **void** printSo()

{

System.***out***.println("Sophomore"+so);

}

}

**class** Junior **extends** UndergraduateStudent

{

**int** j;

**public** **void** printJ()

{

System.***out***.println("Junior"+j);

}

}

**class** Senior **extends** UndergraduateStudent

{

**int** se;

**public** **void** printSe()

{

System.***out***.println("Senior"+se);

}

}

**class** GraduateStudent **extends** Student

{

**int** gs;

**public** **void** printGs()

{

System.***out***.println("GraduateStudent"+gs);

}

}

**class** DoctoralStudent **extends** GraduateStudent

{

**int** ds;

**public** **void** printDs()

{

System.***out***.println("DoctoralStudent"+ds);

}

}

**class** MasterStudent **extends** GraduateStudent

{

**int** ms;

**public** **void** printMs()

{

System.***out***.println("MasterStudent"+ms);

}

}

7)

In Inheritance

If we have a blue print in prepread in earlier ,

Inheritance gave a way to make a same blueprint

If we use methords in this case

We reuse all methrds,

class Calculator{

public static int EXIT\_ON\_CLOSE=3;

public void setSize(int length, int width){

}

public void setTitle(String title){

}

public void setDefaultCloseOperation(int a){

}

public void setVisible(boolean b){

}

}

class Demo{

public static void main(String args[]) {

Calculator c1=new Calculator();

c1.setSize(300,300);

c1.setTitle("Calculator");

c1.setDefaultCloseOperation(Calculator.EXIT\_ON\_CLOSE);

c1.setVisible(true);

}

}

But using Inheritance we can

Use software reusability

import javax.swing.\*;

class Calculator extends JFrame{

//

//

//

}

//Calculator IS-A JFrame

//JFrame -->Super Class

//Calculator-->Sub class

// "extends JFrame"--->Inheritance

class Demo{

public static void main(String args[]) {

Calculator c1=new Calculator();

c1.setSize(300,300);

c1.setTitle("Calculator");

c1.setDefaultCloseOperation(Calculator.EXIT\_ON\_CLOSE);

c1.setVisible(true);

}

}

9)

A,B,C,D,E,F

10)

Cmpile error at

Line 2

And

Line 4

11)

H

I

12)

A  
B(constructor is not a membr)

C

D

E

13)

Compile error at

Line 01-

Line 02

Constructor call must be the first statement in a constructor

14) Compile error

The super keyword can also be used to invoke the parent class constructor.

This case Sub class did not invoke

Super class Constructor

15)

16)Compile error

17) E,F,X,Y,

18)

Constructors are not inherited in a derived class. Therefore, a child's constructor is responsible for calling the parent's constructor:

public Stack()

{

super();

}

By keyword super Java provides a mechanism to call parent's constructor. The above super() is actually a call for ArrayList's constructor. The super must be used in the first line (literally) of a constructor.

If the child class constructor does not call super, the parent's constructor with no arguments will be implicitly called. If parent class implements a constructor with arguments and has no a constructor with no arguments, then the child constructors must explicitly call a parents constructor.

The super reference can also be used to invoke other methods in parent's class. How would you call the ArrayList's toString method from inside the Stack's toString?. Since both methods have the same signature, we say that Stacks's toString *overrides* Book's toString(). We will distinguish between them using this and super keywords. Consider the following code example and trace new B().m1() call:

public class Demo

{

public static void main(String[] args)

{

System.out.println( new B().m1() );

}

}

class A

{

int m1() {return m3();}

int m2() {return this.m3();}

int m3() {return 1;}

}

class B extends A

{

int m1() {return super.m1();}

int m3() {return 4;}

int m4() {return m2();}

}

Here is a trace: B.m1() -> A.m1() -> B.m3() -> 4.

A child class can override any public parent's method that is not defined with the final modifier. Also, classes with final modifier cannot be extended. The Java's String and StringBuffer classes are examples of such classes.

19)

B

20)

21)D

22)

//out put

Inside area for rectangle

Area is45.0

inside are for triangle

Area is40.0

Area of figure is undefine

Area is0.0

23)E

24)

A

C

D

25)

26)A

27)

D

29)

C

D

E

30)