Chapter ❋ INHERITANCE ❋

P55

```
1) Class A
{
    static int j=90;
    int i=12;
    double d=8.3;
}

Class B
{
    static int j=2348;
    int k=67;
    float f = 23.43f;
}

Class Run1
{
    psvm ( )
    {
     Sop("Program starts...");
     A a1 = new A();
     Sop ("i="+a1.i);
     Sop ("d=" + a1.d);
     Sop ("j=" + a1.j);
     Sop ("-----------");
     B b1 = new B();
     Sop ("k=" + b1.k);
     Sop ("f=" + b1.f);
     Sop ("j=" + b1.j);
     Sop ("Program ends...")
    }
}
```

O/P:

```
Program starts...
i = 12
d = 8.3
j = 90
--------
```

1* Inheriting a member of one class to another class is known as Inheritance

2* The class from where members are inherited are known as Super Class / parent class / base class.

3* The class to which members are inherited are known as Sub class / child Class / derived class.

4* A sub class can inherit a super class by using 'extends' keyword

Syntax :-

```
Class SubClassName extends SuperClass
{
    ____
    ____
    ____
}
```

5* The inheritance always happens from super class to SubClass.

6* Only non-static members of SuperClass are inherited to Sub Class.

7* The static member of SuperClass will not be inherited to SubClass because :-

NOTE:

* The static members will be loaded into static pool at the time of Classloading.
* Static members will not be loaded into the Object.

O/p continued...

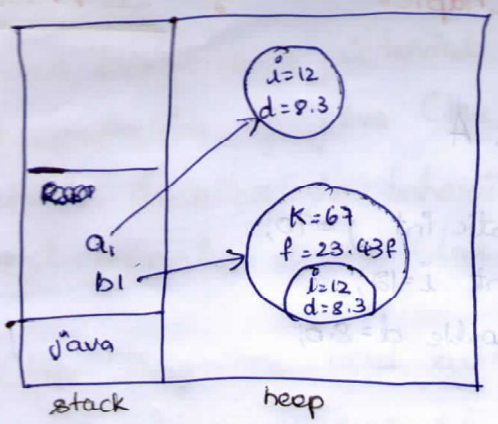
```
K = 67
f = 23.4
j = 2348
Program ends...
```

2) Class A

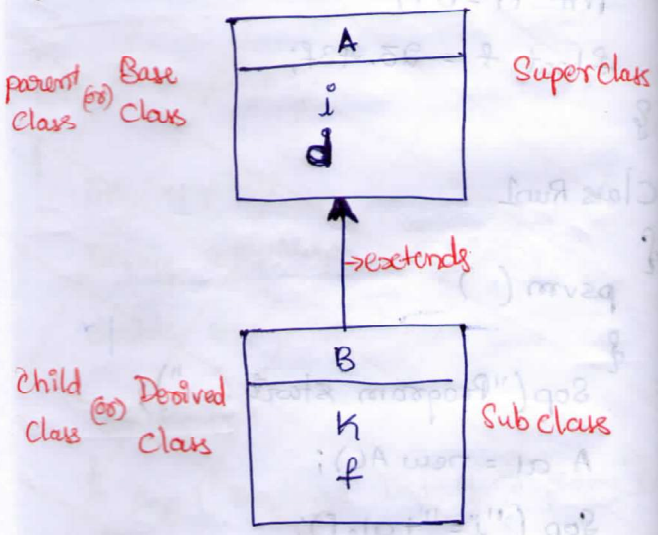
```
{
    int i=12;
    double d=8.3;
}

Class B extends A
{
    int K=67;
    float f=23.43f;
}

Class Run1
{
    psvm ( )
    {
    Sop("Program starts...");
    A a1 = new A();
    Sop("i=" + a1.i);
    Sop("d=" + a1.d);
    Sop("----------");
    B b1 = new B();
    Sop("K=" + b1.K);
    Sop("f=" + b1.f);
    Sop("i=" + b1.i);
    Sop("d=" + b1.d);
    Sop("Program ends...");
    }
}
```

O/p: Program starts...
i = 12
d = 8.3

K = 67
f = 23.43
i = 12
d = 8.3
Program ends...



9*) Pictorial Representation of Inheritance



3) P57

```
class C
{
  int i=78;
}
Class D extends C
{
  double d=12.34;
}
Class E extends D
{
  float f = 324.0f;
}

Class Run2
{
   psvm ( )
   {
     Sop ("Program starts...");
     D d1 = new D( );
     Sop ("i =" + d1.i);
     Sop ("d =" + d1.d);
     Sop ("_ _ _ _ _ _ _ _ _");
     E e1 = new E ( );
     Sop ("i =" + e1.i);
     Sop ("d =" + e1.d);
     Sop ("f =" + e1.f);
     Sop ("program ends...");
   }
}
```

O/P:

```
Program starts...
i = 78
d = 12.34
_ _ _ _ _ _ _ _ _
i = 78
d = 12.34
f = 324.0f
Program ends...
```

Assignment 30-01-2013

P58

```
class Employee
{
    int empID;
    String empName;
    double empSalary;
    Employee()
    {
        Sop ("running default constructor");
        this.empID = 000;
        this.empName = "noname";
        this.empSalary = 0.0;
    }
    Employee(int empID, String empName, double empSalary)
    {
        Sop("running 3 arg constructor");
        this.empID = empID;
        this.empName = empName;
        this.empSalary = empSalary;
    }

    int getEmpID()
    {
        return this.empID;
    }

    String getEmpName()
    {
        return this.empName;
    }
    double getEmpSalary()
    {
        return this.empSalary;
    }
```

O/P:

```
Program start...
running default constructor
Emp ID : 12012
Emp Name : Ramesh
Emp Salary : 23455.87
Program ends...
```

```
void setempID (int empID)
{
    this.empID = empID;
}
void setempSalary(double empSalary)
{
    this.empSalary = empSalary;
}
void setEmpName(String empName)
{
    this.empName = empName;
}
class CreateEmployee
{
    psvm()
    {
        Sop("Program starts...");
        Employee empl = new Employee(12012, "Ramesh", 23455.87);
        Sop("Emp ID:" + empl.getEmpID());
        Sop("Emp Name:" + empl.getEmpName());
        Sop("Emp Salary:" + empl.getEmpSalary());
        Sop("Program ends...");
    }
}
```

31-01-2013 Thursday

Constructor Chain

4) P59

```
class F
{
    F()
    {
      Sop("running F() constructor...");
    }
}
class Gr extends F
{
    Gr()
    { super();
      Sop("running Gr() constructor...");
    }
}
Class H extends Gr
{
    H()
    { super();
      Sop("running H() constructor...");
    }
}
Class Run3
{
  psvm()
  { Sop("Program starts...");
    H h1 = new H();
    Sop("Program ends...");
  }
}
```

O/P: Program starts...
Running F() constructor...
Running Gr() constructor...
Running H() constructor...
Program ends...

1*) In inheritance, when an object of sub class is created, the constructor of sub class calls the constructor of super class.

2*) The super class constructor calls its super class constructor this is known as Constructor Chain

3*) Inheritance happens only if the constructor chain happens

4*) The chain of constructor is implicitly done by compiler by using a statement called super() statement.
The super() statement calls the constructor of super class.

5*) Whenever compiler makes an implicit call to super class, it always calls to default constructor of super class.

6*) If super class doesn't have default constructor then sub class constructor should explicitly calls parametrized constructor of Super Class.

7*) The Super() statement has to be written in the first line of constructor body.
Inside constructor body

5) P60

```
class F
{
   F()
    {
      super(); //calls
      Sop("running F() constructor...");
    }
}
class Gr extends F
{
    Gr(int a)
    {
      Sop("running Gr() constructor...");
    }
}
class  H extends Gr
{
   H()
    {
      super(12);
      Sop("running H() constructor...");
    }
}

Class Run3
{
    psvm ( )
     {
       Sop ("Program starts...");
       H h1 = new H ( );
       Sop ("Program ends...");
     }
}
```

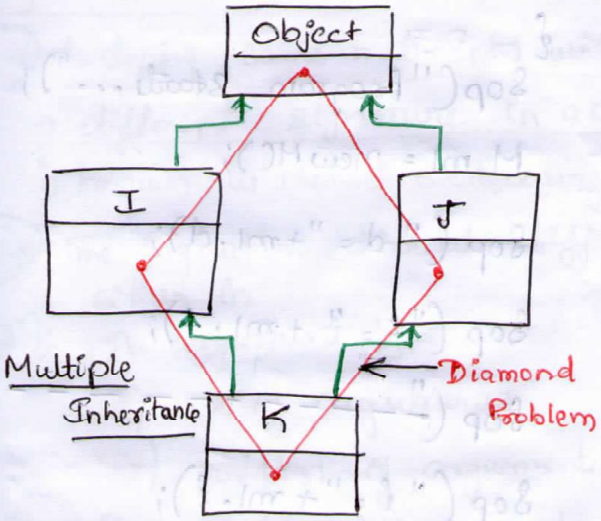
O/p: Program starts...
running F() constructor...
running Gr() constructor...
running H() constructor
Program ends.

we cannot write both this() & super() statement. either one of statement should be written

8*) Every java class should inherit from Object class.

Object class is supermost class in java.

9*) Diamond Problem



Example

```
Class I
{
}
class J
{
}
class K extends I, J
{   //error, multiple inheritance
    not supported in java using Class
}
```

6) P61

```
class L
{
    static int i=12;
           double d=23.45;
}
class M extends L
{
    float f=12.34f;
}
class Run5
{
    psvm( )
    {
     Sop("Program starts...");
     M m1 = new M();
     Sop (" d = "+m1.d);
     Sop ("f = " +m1.f);
     Sop ("----------");
     Sop ("i = "+m1.i);
     Sop ("Program ends...");
    }
}
```

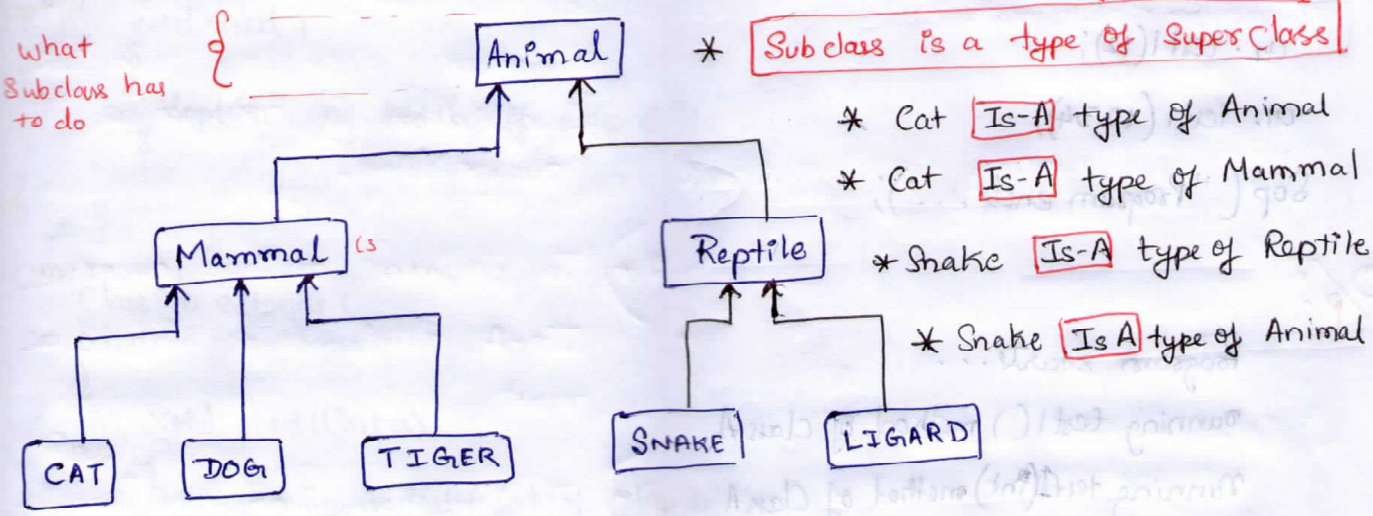
O/P:

10) Inheriting more than one class members at a time is known as multiple inheritance.

Java doesn't support multiple inheritance using class because subclass constructor cannot make call to more than one super class. Also Object class member cannot be inherited to single subclass in more than one path.

01-02-2013 | Friday

* method - behaviour
* variables - fields



* Subclass is a type of Superclass
* Cat Is-A type of Animal
* Cat Is-A type of Mammal
* Snake Is-A type of Reptile
* Snake Is A type of Animal

P68

METHODS | OverLoading

7)

```
class A
{
    void test1()
    {
        Sop("running test1() method of class A");
    }

    void test1(int a)
    {
        Sop("running test1(int) method of Class A");
    }

    void test1(double a)
    {
        Sop("running test1(double) method of Class A");
    }
}

Class Run1
{
    psvm()
    {
        Sop("Program starts...");
```

1 * developing same methods() with a different signature in a class is known as Method overloading.

2 * The signature should differ either in
- 2.1) argument type
- 2.2) No of arguments
- 2.3) position of arguments.

3 * Both static methods and non-static methods of a class can be overloaded in the class.

4 * A super class method can be overloaded in sub class

5 * A method overloading should be done whenever same operation has to implemented based on arguments.

6 * The jvm executes the method based on arguments value.

```
A a1 = new A();
a1.test1();
a1.test1(12);
a1.test1(12.34);
Sop("Program ends...");
```

O/P:

```
Program starts...
running test1() method of class A
running test1(int) method of ClassA
running test1(double) method of ClassA
Program ends...
```

---

8) Class B P63

```
{
  static void test1()
  {
    Sop("running test1() method of classA");
  }
  Static void test1(int a)
  {
    Sop("running test1(int) method of ClassA");
  }
  Static void test1(double a)
  {
    Sop("running test1(double) method of ClassA");
  }
```

15*

26 *

9) P64

```
Class C
{
   void test1()
   {
     Sop("running test1() of class C");
   }
}
Class D extends C
{
    void test1(int a)
    {
      Sop("running test1(int of Class D");
    }
}

Class Run3
{
    psvm()
   {
    Sop ("Program starts...");
    D d1 = new D();
     d1. test1(); // inherited member
     d1. test1(123); // derived member
     Sop("Program ends...");
    }
}
```

O/P:

```
Program starts...
running test1() of class C...
running test1(int) of class D...
Program ends...
```

10)

```
class C
{ //overridden method
    void test1()
    {
      Sop("running test1() of class C");
    }
}

class D extends C
{
  //overridded method
   void test1()
   {
     Sop("running test1() of class D");
   }

class Run3
{
    psvm()
    {
      Sop("Program start ...");

      D d1 = new D();
      d1.test1();

      Sop ("Program ends ...");
    }
}
```

O/P:

```
Program starts ...
running test1() of class D
Program ends ...
```

1.* Inheriting the behavior of Superclass and changing the behavior according subclass Specification is known as overriding.

2.* To perform method overriding Is-A relationship should be must.

3.* The method signature in the subclass should be same as Superclass type.

4.* Sub class should provide a diff implementation.

5.* whenever an object of Subclass is created for overridded method sub class implementation will be available.

6.* We cannot override static methods of a super class because it will not inherited to sub class object.

7.* The super class method which is overridded in Subclass is called as overridden method the same method in a sub class is called as overridded method.

04-02-2013 | MONDAY

abstractclasses

ABSTRACT METHOD

P66

```
abstract class A
{
  static void sample1()
  {  // concrete method
   Sop("running sample1()...");
  }
  abstract void sample2();//abstract method
}
class Run1
{
  pevrm(String[] args)
  {
   Sop("program starts...");
   A.sample1();
   Sop("program ends...");
  }
}
```

O/P:
Program starts...
running sample1()
Program ends...

Abstract Classes / Method

1*) A method developed with signature & body is known as concrete method (or) Complete method.

2*) Any method developed without body is known as abstract method.

3*) The abstract method should be declared with a keyword abstract.

4*) If a class contains atleast one abstract method, then the class should be declared as abstract.

5*) Inside an abstract class, we can develop both abstract & concrete method.

6*) If a class is declared as abstract it's not mandatory to develop abstract method.

7*) We cannot create an instance of abstract class, hence we cannot access non-static members of abstract class.

8*) The static members of an abstract class can be refered using class name.

9*) We can develop empty abstract class.

P67

2)

```
abstract Class B
{
    abstract void sample1();
}
class C extends B
{
    void sample1()
    {
        Sop("Sample1() implemented in class C");
    }
}
class Run2
{
    psvm (String[] args)
    {
        Sop("Program starts...");
        C c1 = new C();
        c1.sample1();
        Sop("Program ends...");
    }
}
```

O/p:-

```
Program starts...
Sample1() implemented in class C
Program ends...
```

10*) A subclass inheriting an abstract class should override all the abstract methods of the abstract class, otherwise the subclass should be declared as abstract.

11*) An abstract method of an abstract class, can get the body in any of the Sub class level.

3) P68

```
abstract class D
{
   abstract void demo1();
   abstract void demo2();
}
abstract class E extends D
{
   void demo1()
   { Sop("demo1() implemented in class E");
   }
}
class F extends E
{
      void demo2()
      {
        Sop("demo2() implemented in class F");
      }
}
class Run3
{
   psvm(String[] args)
   {
    Sop ("Program starts...");
      F f1 = new F();
      f1. demo1();
      f2. demo2();
      Sop ("Program ends...");
   }
}
```

Op;
```
Program starts...
demo1() implemented in class E
demo2() implemented in class F
Program ends...
```

4) abstract class G1

```
{
    void sample1()
    {
     Sop("sample1() implemented in class G1");
    }
}
class H extends G1
{
   H()
   {
     Sop("running H() constouctor");
   }
}
Class Run4
{
   psvm(String[] args)
   {
    Sop("Program starts...");
    H h1 = new H();
    h1.sample1();
    Sop("Program ends...");
   }
}
```

O/p: Program starts...

running H() constructor
Sample1() implemented in class G1
Program ends...

05-02-2013 | TUESDAY

P70

5)
```
class I
{
  final void test1()
  {
    sop("running test1() of class I");
  }
}
class J extends I
{
  //error, cannot override final methods of super class
  void test1()
  {
    sop("running test1() of class J");
  }
}
```

6) final abstract class I
// error: abstact final illegal combination

7)
```
abstract class I
{
  final abstract void test1();
}
```

8)
```
abstract class I
{
  abstract static void test1();
}
```

9)
```
final class I
{
}
Class J esctends I
```

12*) A method can be declared as final, final methods behavior cannot be changed, means final methods cannot be overrided, final methods can be inherited to sub class from super class but not overrided.

13*) A class can be declared as final, such classes cannot have sub class.

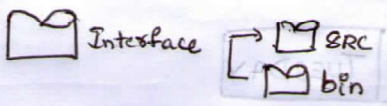
14*) We concreate an instance of final class & execute the members of final class.

NOTE:

15*) final abstract combination is illegal because as per abstract the method() should be overridded in subclass and as per final method() should not be overridded.

16*) abstract static is an illegal combination because static methods cannot be overridded.

INTERFACE



P71

```
interface Sample1
{
  void test1();
}
class A implements Sample1
{
  public void test1()
  {
    sop("test1() implemented in class A");
  }
}

class Run1
{
  psvm(String[] args)
  {
  sop("Program starts...");
  A a1 = new A();
  a1.test1();
  sop("Program ends...");
  }
}
```

O/P:

```
Program starts...
test1() implemented in Class A
Program ends...
```

1*) An interface is one of the java type definition block which is 100% abstract.

2*) All interface methods are by default abstract and public

3*) An interface method need not to be declared as abstract bec by default all methods are abstract.

4*) We cannot develop static methods inside interface.

5*) An interface methods cannot be declared as final.

6*) Interface variable has to be initialized at the time of declaration.

7*) By default the interface itself is a abstract.

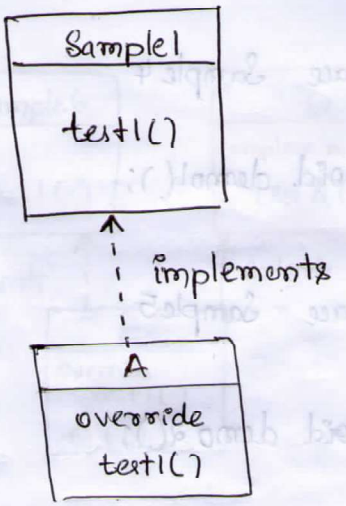
8*) Compiler generates Class file for interface definition block.

9*) An interface methods can get body or implementations in Sub Classes.

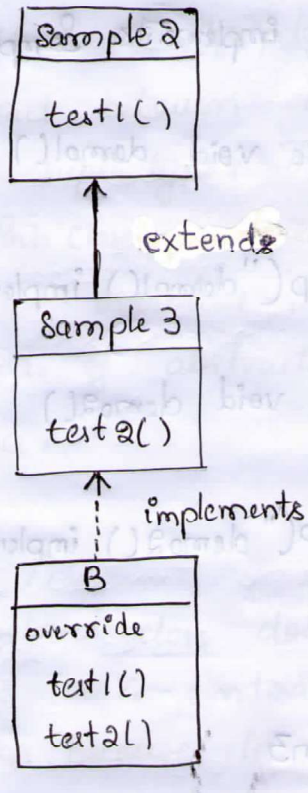
2) P72

```
interface Sample2
{
    void test1();
}
interface Sample3 extends Sample2
{
    void test2();
}
class B implements Sample3
{
  public void test1()
   {
     Sop("test1() implemented in class B");
   }
  public void test2()
   {
     Sop("test2() implemented in ClassB");
   }
}
class Run2
{
  psvm (String[] args)
   {
     Sop ("Program starts...");
     B b1 = new B()
     b1.test1();
     b1.test2();
     Sop ("Program ends...");
   }
}
```

Ex 1)

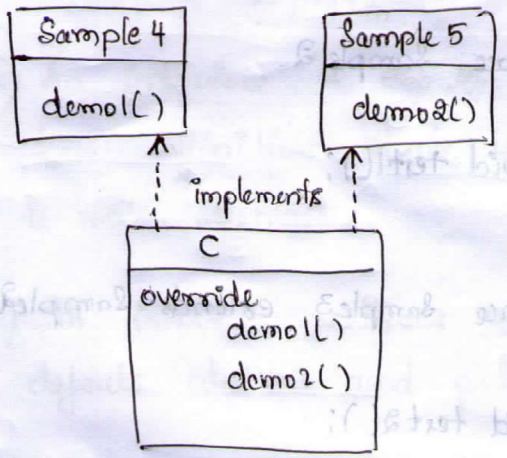


Ex 2)



3) P73

```
interface Sample4
{
    void demo1();
}
interface Sample5
{
    void demo2();
}

class C implements Sample4, Sample5
{
  public void demo1()
  {
    Sop("demo1() implemented in ClassC");
  }
  public void demo2()
  {
    Sop("demo2() implemented in ClassC");
  }
}

class Run3
{
  psvm(String[] args)
  {
    Sop("Program starts...");
    C c1 = new C();
    c1.demo1();
    c2.demo2();
    Sop("Program ends...");
  }
}
```



10*) Sub class inherting an interface should implement all the abstract methods of interface, otherwise sub class becomes abstract.

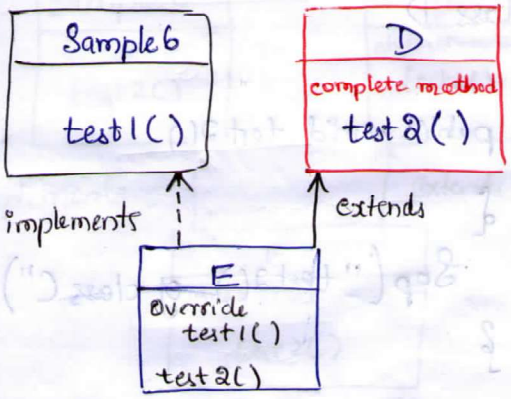
11*) A subclass can inherite more than one interface, in such case the sub-class should overwrite all the interface methods.

12*) A class can inherit from both class and interface.

If the super class is abstract then sub-class should override both abstract class and interface methods().

4) P74

```
Interface Sample6
{
   void test1();
}

class D
{
   void test2()
   {
      Sop("running test2() method");
   }
}

class E extends D implements Sample6
{
   public void test1()
   {
      Sop("test1() implemented in Class E");
   }

   void test2()
   {
      Sop("running test2() in Class E");
   }
}

class Run4
{
   psvm(String[] args)
   {
      Sop("Program starts...");
      E e1 = new E();
      e1.test1();
      e1.test2();
      Sop("Program ends...");
   }
}
```



13*) An abstract class as well as interface describes/defines the contract between the sub class and super type. The contract is Sub Class can exsist only if it overrides the abstract methods of abstract class or Interface.

14*) Abstract class doesn't enforce 100% contract with sub-class because in abstract Class we can develop concrete methods.

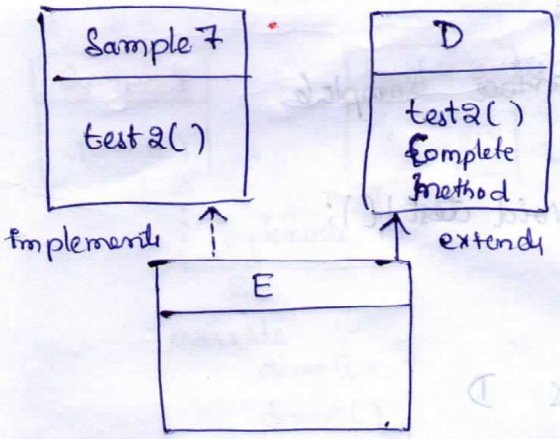
15*) Interface enforces 100% contract with sub classes because inside interface we cannot develop Concrete Methods().

O/p: Program starts...
test1() implemented in Class E
running test2() in class E
Program ends...

5) P75

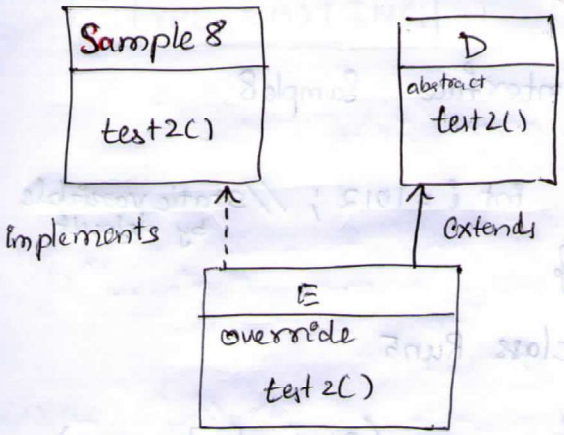
```
class D
{
  public void test2()
  {
    Sop("test2() of class C");
  }
}

interface Sample7
{
  void test2();
}

class E extend D implents Sample7
{
}

class Run4
{
  public vsm ( )
  {
    Sop("Program starts...");
    E e1 = new E();
    e1. test2();
    Sop ("Program ends...");
  }
}
```

O/p: Program starts...
test2() of class C
Program ends...



16*) While developing an application if sub class behavior changes, then those behaviors are defined as abstract in interface, so that the sub class can implement according to sub-class specification.

6)
```
abstract class D
{
  abstract void test2();
}

interface Sample8
{
   void test2();
}

class E extends D implements Sample6
{
   public void test2()
   {
      Sop("test2() implemented in class E");
   }
}

class Run4
{
   psvm()
   {
      Sop("Program starts.....");
      E e1 = new E();
      e1.test2();
      Sop("Program ends...");
   }
}
```



O/p:
```
Program starts....
test2() implemented in class E
Program ends...
```

P77

7)

```
interface Sample8
{
    int i = 1012; //static variable by default
}
class Run5
{
    psvm (String[] args)
    {
        Sop("Program starts...");
        Sop("i = " + Sample8.i);
        Sample8.i = 1227; //error;
        // interface variable are final
        // can't reassign.
        Sop("i = " + Sample8.i);
        Sop("Program ends...");
    }
}
```

17*) All interface variable by default final and static

---

1. * Java understands only Homogenous

LHS = RHS

- int i = 10;
- double d = 21.66;
- boolean b = true;

1) int i;
   type ↑ ← read
   how to read? i is of integer type
2) double d; d is of double type
   type ↑
3) boolean b; b is of boolean type

2* hetrogenous assignment statements

LHS ≠ RHS

- int k = 21.66;
- double b = 10;

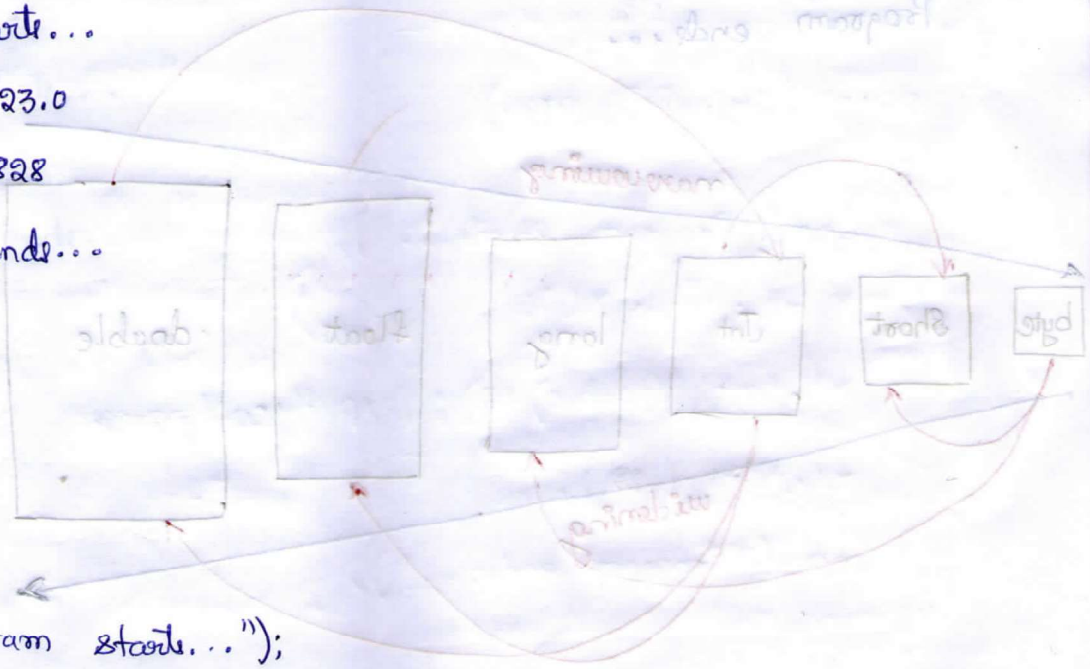
06-02-2013 | WEDNESDAY

TYPE CASTING

p78

```
1) class Run1
   {
     psvm( )
     {
       Sop("Program starts...");
       double d = (double)123;//Widening
       Sop("d value: "+d);
       int i = (int) 828.723;//narrowing
       Sop("i value: " + i);
     }
   }
```

O/p: Program starts...
d value: 123.0
i value : 828
Program ends...



p79

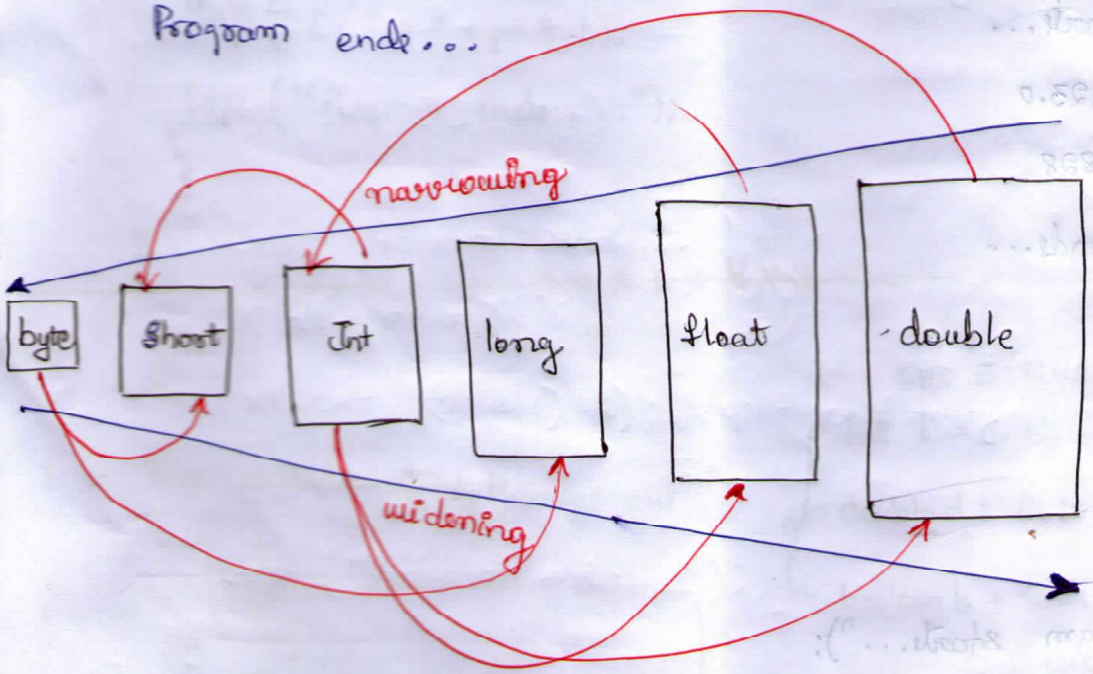
```
2) class Run2
   {
     psvm( )
     {
       Sop("Program starts...");
       int k=123;
       double d = k; // auto widening
       Sop("d value: " + d);
       double b = 876.82;
       int i = (int) b; // explicit narrowing
       Sop("i value: "+ i);
       Sop("Program ends...");
     }
   }
```

O/p: Program starts...
d value: 123.0
i value: 876
Program ends...

P80

3)
```
class Run2
{
    psvm (String[ ] args)
    {
    Sop ("Program starts...");
    int i = 89;
    short s = (short) (float) (int) (double) (short) i;
    Sop ("s = "+s);
    Sop ("Program ends...");
    }
}
```

O/P: Program starts...
s = 89
Program ends...



datatype Variablename = (datatype) value;

PSI

4) class Run4

```
{
  psvm( )
  {
    Sop("Program starts...");
    demo1((int)12.23); //explicit narrowing
    Sop("Program ends...");
  }
  static int demo1(int a)
  {
    double res = a*2; //auto widening
    Sop("res= "+res);
    return (int) res; //explicit narrowing
  }
}
```

O/P:

```
Program starts...
res = 24.0
Program ends...
```

P82

5)
```
class X
{
    void test1(double d)
    {
      Sop ("running test1(double) of class X");
      Sop ("d = " + d);
    }
    void test1( int a)
    {
      Sop ("running test1(int) of class X");
      Sop ("a = " + a);
    }
}
class Run3
{
    psvm (String[] args)
    {
      Sop ("Program starts...");
      X x1 = new X();
      x1.test1 ((double) 12); //explitly widenning
      x1.test1 (12);
      Sop ("Program ends...");
    }
}
```

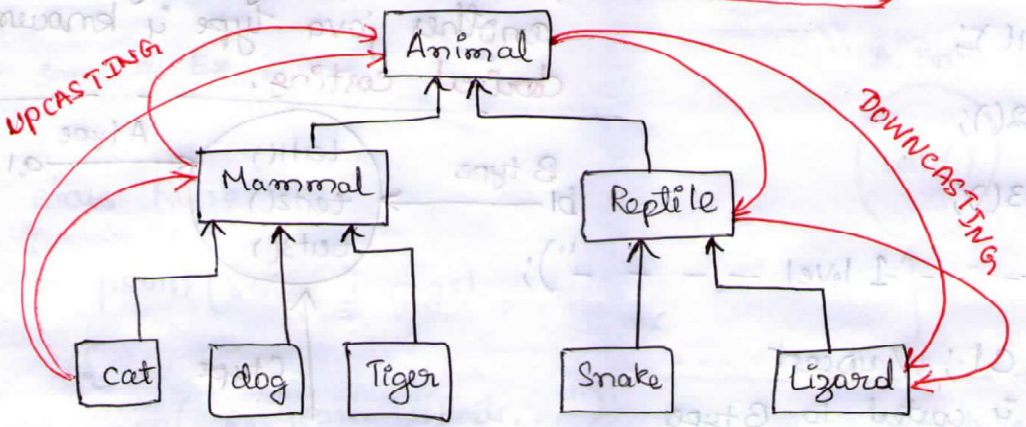
Program Start

1*) In this example class X contain overloaded test1(), one of the method takes doubletype argument other one integertype argument, while invoking the method if we pass integer value jvm executes method which is defined with integer argument type because preference will be given for same type.

If we have to call doubletype method, then we have to explicitly widening the integer value.

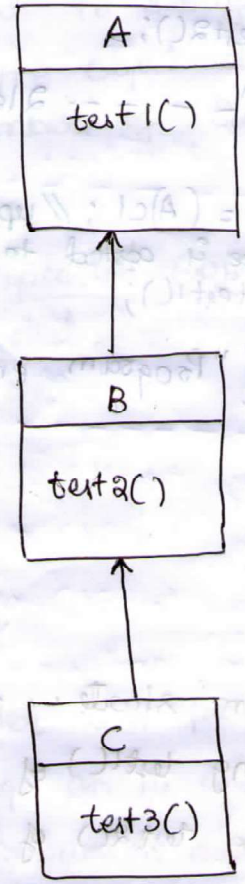
07-02-2013 THURSDAY

Derieved TYPECASTING



P83

```
1) class A
   {
     void test1()
     {
       Sop("running test1() of class A");
     }
   }
   class B extends A
   {
     void test2()
     {
       Sop("running test2() of class B");
     }
   }
   class C extends B
   {
     void test3()
     {
       Sop("running test3() of class C");
     }
   }
   class Run5
   {
     psvm(String[] args)
     {
       Sop("Program starts...");
```

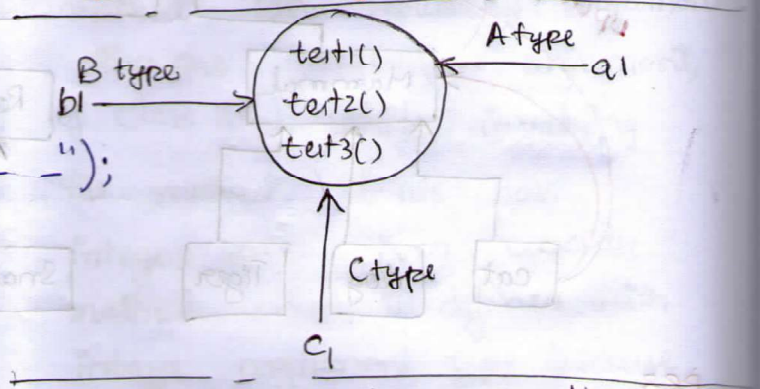


```
C c1 = new C();
c1.test1();
c1.test2();
c1.test3();
Sop("- - - - - 1 level - - - - -");
B b1 = (B) c1; // upcast
// C type is casted to B type
b1.test1();
b2.test2();
Sop("- - - - - 2 level - - - - -");
A a1 = (A)c1; // upcast
// C type is casted to A type.
a1.test1();
Sop("Program ends ...");
}
}
```

O/p:

```
Program starts...
running test1() of A type
running test2() of B type
running test3() of C type
- - - - - 1st level - - - -
running test1() of A type
running test2() of B type
- - - - - - 2nd level - - -
running test1() of A type
Program ends ...
```

1*) Converting a java type to another java type is known as derived casting.



2)* In derived casting there are 2 types
*) Upcast
*) Downcast

3*) Converting an object of sub-type to Super type is known as Upcasting.

When a object is upcasted, the object should not exhibit behavior of sub class.

4*) In other words a super type reference, pointing to sub class object is known as Upcasting.

5*) Converting Super type object to a sub type is known as Downcasting

6*) A downcasting of new Super type compiles successfully by throws an exception by name ClassCastException during execution,

P84

2)

Same as Ex 1:-

```
class Runb
{
    psvm (String[] args)
    {
        Sop("program starts...");
        A a1 = new A();
        a1.test1();
        Sop ("-- -- -- -- -- --");
        B b1 =(B) a1; // downcast
// A-type is casted to B type
// exception occurs, bec downcast
// is not possible in this way.
        Sop("Program ends...");
    }
}
```

because super type object cannot be casted to sub type since Super type object doesn't have the members of Sub type.

7*) Only upcasted Object can be downcasted.

~~8*~~)

P85

3) Same as Ex 1.

```
class Run7
{
    psvm( )
    {
        Sop("Program starts..");
        C c1 = new C();
        Sop("----------");
        A a1 = c1; //auto upcast
        a1.test1();
        Sop("----------");
        B b2 =(B) a1; // downcast
        b2.test1();
        b2.test2();
```

8*) Compiler can perform upcasting on its own, hence it is known as auto upcast.

9*) Downcasting should be explicitly done in the program

```
Sop("----------");
C c2 =(C) a1; // downcast
c2.test1();
c2.test2();
c2.test3();
Sop("Program ends...");
}
}
```

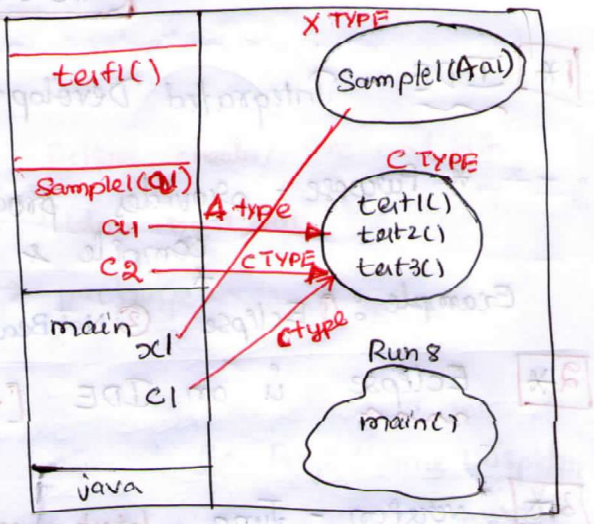
O/P:-

```
Program starts...
----------
running test1() of class A
----------
running test1() of class A
running test2() of class B
----------
running test1() of class A
running test2() of class B
running test3() of class C
Program ends...
```

P86

4) Same as Ex 1

```
:
:

class X
{
    void sample1(A a1)
    {
        // How to access members of C here
        C c2 = (C) a1;
        c2.test1();
        c2.test2();
        c3.test3();
    }
}

class Run8
{
    psvm(String[] args)
    {
        Sop("Program starts...");
        X x1 = new X();
        C c1 = new C();
        x1.sample1(c1); //auto upcast

        Sop("Program ends...");
    }
}
```



O/P:

```
Program starts...
running test1() of class A
running test2() of class B
running test3() of class C
Program ends...
```

ECLIPSE

1* IDE - Integrated Development Environment

* Purpose - provides programming environment [edit, compile & execute programmes]

Example : ① Eclipse ② Net Beans ③ JDeveloper ④ Jcreater ⑤ IntelliJidea

2* Eclipse is an IDE [majority of industry uses]

3* version - Juno - latest version
Helios
Indigo

4* Steps to download :- 1. eclipse.org/downloads

2. Eclipse IDE for Java Developers [ver Juno]

3. Download zip file [32 bit or 64 bit]

4. unzip file

5. Click on eclipse icon → Workspace Launcher
create a folder wherein all projects are stored
Set the path to Created folder, click on next.

Workspace is a folder

5* How to use Eclipse :- Heirarchy

(Create) Project
↳ Package
↳ classes

5.1*1.) Create Project → File menu → new → project → Select Java project

08-02-2013 FRIDAY

5.1.a * Project Name : Sample-proj → finish

5.1.b * File menu → new → Sample-proj - Eclipse creates src and bin folders by default.

5.2 *) Right Click on Sample_proj → new → package.

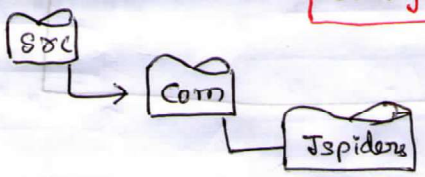
convention to write PackageName.

WWW.ProjectName.com

Ex: ProjectName 1. Jspiders
2. gmail

i.e package Name :- com. ~~project~~ Company Name . ProjectName

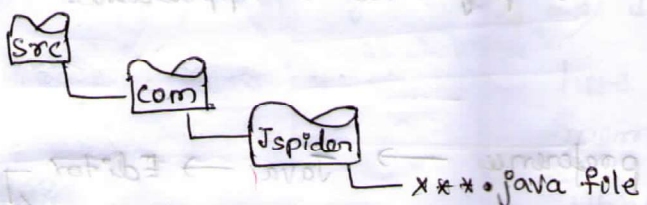
com.Jspiders.Sample



5.3 *) Right Click on Com.Jspiders → new → class

Enter ClassName &

☑ check public static void main (String[] args)



5.4) Write java program save javafile 1) click on Run icon.
output below in view console. 2) right click → run as → Java program

5.5) Shortcut Ctrl + Shift + L for eclipse

Sop shortcut syso + [ctrl + spacebar]

type main → ctrl + space

6 * Layout of Eclipse

6.1. Editor [Type in java program

6.2. Views [console, package list etc]

Window menu → Show view → * console [Comes by default, when run java file]
* project Explorer
* Task list

6.3* Java perspective, Java icon

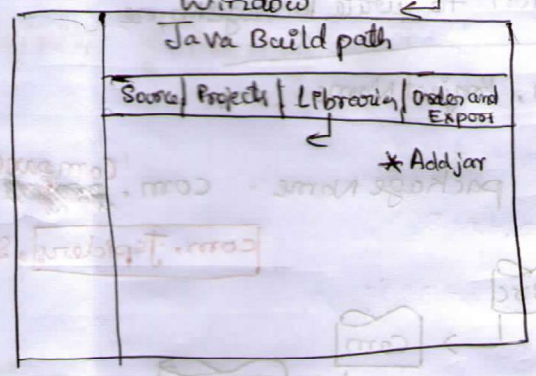
Outline view → display methods() in class

- private
- public
- default
- Yellow protected

7* Import from existing project into current project.

Configure Java path

right click Sample_Proj → Build path → Java Build Path.



8* Change look, appearance, font size etc

Window menu → preference → Appearance

9* Auto Fill in eclipse

Window menu → preference → Java → Editor → Typing

Automatically insert at correct position

- ☑ Semi colon
- ☑ Braces

P87

```
1) class A
   {
      void test1()
      {
        Sop("running test1() of class A");
      }
   }
   class B extends A
   {
      void test1()
      {
        Sop("running test1() of class B");
      }
   }
   class C extends B
   {
      void test1()
      {
        Sop("running test1() of class C");
      }
   }
   class X
   {
      void sample1(A a1)
      {
        a1.test1();
      }
   }
   class Run9
   {
      psvm(String[] args)
      {
        Sop("program starts...");
        X x1 = new X();
        x1.sample1(new A());
        x1.sample1(new B());
        x1.sample1(new C());
```

POLYMORPHISM

1*) A method declared with reference type argument can hold any type of object which is a Sub class to the reference type.

Example :- If a method() takes an argument of A-type, while invoking that method, we can pass any object which is derived from A class type.

While passing a subclass Object, the compiler does auto upcast to the argument type.

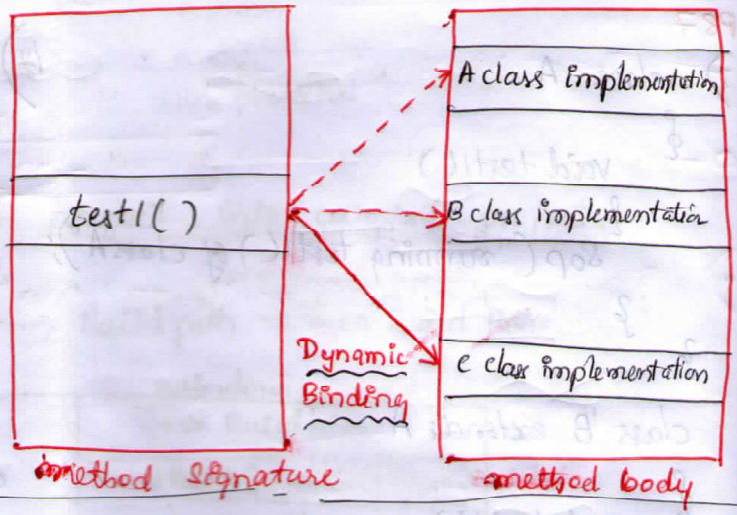
2*) Inside the method if we have to refer the sub class members using upcasted Object, then that Object should be downcasted to the Sub class type.

Sop ("program ends..."); Run Time Polymorphism

}

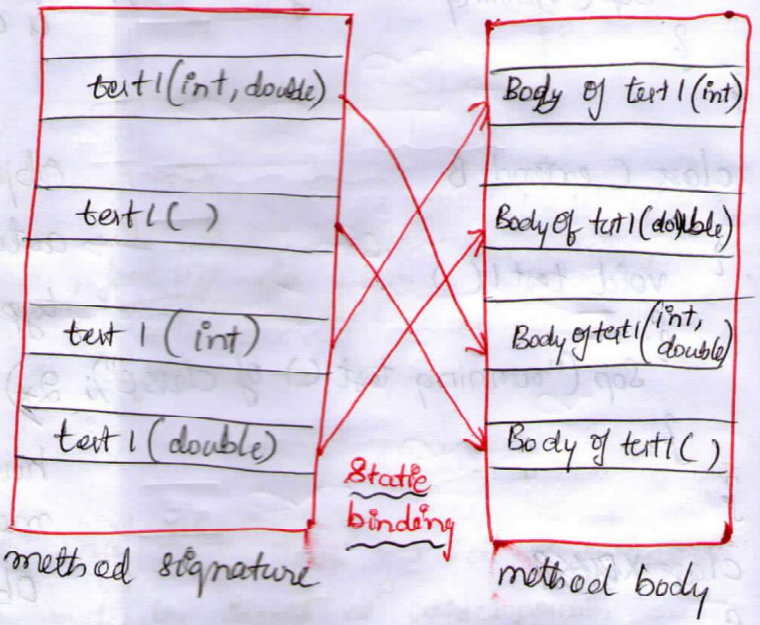
O/P:

Program starts...
running test1() of class A
running test1() of class B
running test1() of class C
Program ends...



Compile time polymorphism:

```
class A
{
    test1()
    {
    }

    test1(int a)
    {
    }

    test1(double d)
    {
    }

    test1(int a, double d)
    {
    }
```



1*) An object behaving differently in a diff condition is known as polymorphism.

2*) Two types of polymorphism is supported in java
1* Compile time polymorphism
2* Run time polymorphism

3*) In Compile time polymorphism method behavior is binded to the method signature at the time of compilation by compiler. This binding is static binding because once binded it cannot change throughout the program. This is also known as Static Polymorphism

4*) Examples of compile time polymorphism is method Overloading & constructor overloading.

5*) In Run time polymorphism the method behavior is binded to method signature at the execution time done by jvm. The binding happens based on object created. The binding can be changed in the program execution, hence it is known as dynamic binding. This is also known as Dynamic polymorphism

6*) Example : method overriding.

To achieve runtime polymorphism following condition should be satisfied.

1* Inheritance (or) Is-A relationship

2* Method overriding

3* Upcast

11-02-2013 | MONDAY

execute command: java packagename.javafilename.

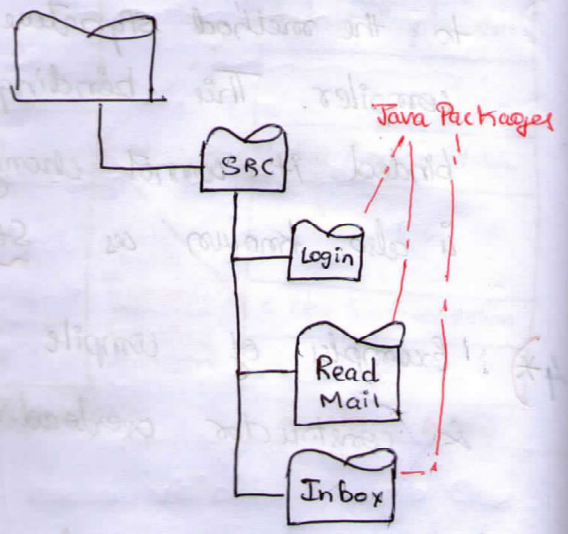
PACKAGES

P88

1)
```
package login; // package declaration

class A
{
  psvm (String[] args)
  {
    Sop("running class A from login");
  }
}

class B
{
  psvm (String[] args)
  {
    Sop ("running classB from login");
  }
}
```

O/p:

2)
```
package inbox; // package declaration

class C
{
  psvm (String[] args)
  {
    Sop("running class C from inbox");
  }
}

class D
{
  psvm (String[] args)
  {
    Sop ("running class D from inbox");
  }
}
```



1* While developing java classes the java classes are grouped into packages.

2* Packages are developed mainly to organise java classes and for reusability purpose.

* ~~Referring a jar class name~~

3* A class associated with a package should be executed by using fully qualified class name

4* Referring a java class along with package name is known as Fully qualified class

5* A class of one package can be referred from other package either by fully qualified classname (or) by importing the package.

3) P89

```
package login;
class A
{
    int i=120;          // access privelage : default
    psvm(String[] args)
    {
      Sop("Program starts...");
      A a1 = new A();
      Sop("i="+a1.i);
      Sop("Program ends...");
    }
}

class B
{
    psvm(String[] args)
    {
      Sop("Program starts...");
      A a1=new A();
      a1.i=234;
      Sop("i="+a1.i);
      Sop("Program ends...");
    }
}
```

4) P90

```
A* private int i=120;  //access privalage: private
```

O/p: class B
error

6* Import statement should be written after package declaration statement

7* We can import one class at a time (or) all classes of the package.

5) P91

```
package login;
public class A
{
    public int i=120;
    psvm ( )
    {
      Sop("program starts...");
      A a1 = new A();
      Sop ("i = "+a1.i);
      Sop ("Program ends...");
    }
}
```

```
package inbox;
public class C
{
    psvm ( )
    {
      Sop (" Program starts...");
      login.A a1 = new login.A();
       a1.i = 785;
      Sop ("i = "+a1.i);
      Sop ("Program ends...");
    }
}
```

Access Specifications

1* The access specifiers defines visibility

2* Java supports 4 types
1) Private
2) default
3) protected
4) Public

3* Private: has a visibility upto Class level members
They cannot be referred (or) accessed outside the class.

4* default members: has a visibility upto package level. The default members can be referred from any class within the package.

5* public members has a wider visibility. Public members of an class can be referred from outside the package or within the package.

Public members of an Class can be referred from outside package only if the class is public.

6* Protected members has visibility upto package level, it can be referred outside package through inheritance, once a protected members are inherited behaves like a private in subclass

12-02-2013 | Tuesday

P92

6) Same as previous example
package login;
public class A

```
package inbox;
import login.A;
public class C extends A
{   psvm( )
  {  Sop("Program starts...");
     C c1 = new C();
     c1.i = 7665;
     Sop("i= "+c1.i);
     Sop("Program ends...");
  }
}
class D extends C
{
   psvm(String[] args)
   {
     Sop("Program starts...");
     D d1 = new D();
      d1.i = 5465;
      Sop("i = "+d1.i);
      Sop("Program ends...");
   }
}
```

7) An outer class should be either public or default

8) Inner class can have any of the four access specifiers.

9) An outer class cannot be declared as private bec it cannot be used or referred anywhere.

10) An outer class cannot be declared as protected bec protected members can be referred only through inheritance, class will not be inherited, only members of class will be inherited.

11) A javafile can have only one public class, in such case the filename should be same as public class.

12) A private variable of a class can be referred from another class by using getters & setters method. Both getters & setters method should be public.

7)

```
package login;
class C
{
  private C()
  {
    Sop("running C() constructor");
  }
  psvm(String[] args)
  {
    Sop("Program starts...");
    C c1 = new C();
    Sop("Program ends...");
  }
}

class Run1
{
  psvm(String[] args)
  {
    Sop("program starts...");
    C c1 = new C();
  // error, Cant refer private constructor from another class
    Sop("Program ends...");
  }
}
```

8)

* A java class which has following is known as Java Bean class

1) public constructors
2) private variables
3) public getters & setters method.

* java beans are used in development of enterprise application

Getters Method & Setters Method

8) P94

```
package login;
class Student
{
   private int stID=191;
   private double stMarks = 12.58;
   // getters Method
   public int getStID()
   {
      return stID;
   }
   public double getStMarks()
   {
      return stMarks;
   }
   // Setters Method
   public void setStID(int stID)
   {
      this.stID = stID;
   }
   public void setStMarks(double stMarks)
   {
      this.stMarks = stMarks;
   }
}
class Run2
{
   psvm (String[] args)
   {
      Sop ("Program starts...");
      Student st1 = new Student();
      Sop ("Student ID: " + st1.getStID());
      Sop ("Student Marks: " + st1.getStMarks());
      Sop ("Program ends...");
   }
}
```

Can u refer a private members of a class?

Yes, by getters, setters method() and set the methods as public.

* Getters method returns private variable value.
* Setters method initialize/set the private variable value.
* ~~#~~ Gossa

O/p: Program starts...
Student ID: 191
Student Marks: 12.58
Program ends...

Singleton Class

q) P95

```
package login;
class Gr
{
  // Singleton class
private static Gr g1 = null;
  private Gr()
  {
    Sop("running Gr() constructor");
  }
  public static Gr getInstance()
  {
    if(g1==null)
    {
      Sop("Creating an instance of Gr");
      g1 = new Gr();
    }
    return g1;
  }
  void sample1()
  {
    Sop("running sample1() of Gr Class");
  }
}
class Run3
{
  psvm(String[] args)
  {
    Sop("Program starts...");
    // Get Instance of class Gr
    Gr g2 = Gr.getInstance();
    g2.sample1();
    Gr g3 = Gr.getInstance();
    g3.sample1();
    Sop("Program ends...");
  }
}
```

30*

1* Any java class which allows to create only one instance is known as Singleton Class.

2* A java class can be qualified as a singleton class, if it meets following criterias.

2.1* The constructor should be private

2.2* The class should have a static public method which returns an instance of the class. The method should return only one instance whenever it is invoked.

3* A class should have referrence variable of same type which has to be private.

O/p: Program starts...
creating an instance of Gr
running Gr() constructor
running sample1 of class Gr
running Sample of class Gr
Program ends...

OBJECT CLASS

13-02-2013 | WEDNESDAY

30*
```
/*
    Gr g2 = Gr.getInstance();
    Gr g3 = Gr.getInstance();
    Gr g4 = Gr.getInstance();
    Gr g5 = Gr.getInstance();
*/
```

System class

1*) System is a class available in java.lang package.

2*) java.lang package is imported to every java class by default, hence no need to import explicitly

pg6
10)
```
package com.jspiders.demo;
class Department;
{
 static Student st = new Student(); // st is type of Student
}
class Student
{
  int stID = 1209;
  double stMarks = 81.22;
  psvm (String[] args)
  {
     System.out.println("Student ID: "+stID);
     Sop("Student Marks: "+stMarks);
  }
}
public class Demo1
  {  psvm (String[] args)
     {
       Sop("Program starts...");
       Department.st.dispStDetails();
       System.out.println("program ends...");
     }
  }
```

3*) System class contains two static reference variables which are final

3.1 * in is a type of InputStream.

3.2 * out is a type of PrintStream

4*) The PrintStream Object contains several methods used to print characters on the Standard Output device /Monitors

5*) InputStream contains members or methods used read from Standard Input Device

System.out.println means System is class, out is a static reference variable of type PrintStream and println is a member of PrintStream Object."

toString()

p97 -

ii)
```
package com.jspiders.objectclassdemo;
class A
{

}
public class Demo1
{
  psvm(String[] args)
  {
    sop("Program starts...");
    A a1 = new A();
    // String s1 = a1.toString();
    sop(a1.toString()); // (explicitly call to toString)
    sop("----------");
    A a2 = new A();
    sop(a2.toString());
    sop("program ends...");
  }
}
```

O/P:-

```
Program starts...
com.jspiders.objectclassdemo.A@1a16869
----------
com.jspiders.objectclassdemo.A@1e278b9
Program ends...
```

1* Object class is the supermost class in java where every class has to inherit member of object class.

2* This class is available in package java.lang.

3* The toString() of object class is public method of return type String.

Whenever this method is invoked, the method returns String representation of object.

4* The string representation is in format of

fullyqualified classname@hexadecimal address.

Whenever a reference variable is printed the toString() method is implicitly invoked, hence return value of toString() method is displayed.

P98

12) package com.jspiders.objectclassdemo;

```
public class Demo2
{
    psvm(String[] args)
    {
        Sop("program starts...");
        A a1 = new A();
        Sop("a1") // implicit call to toString()
        Sop("-- ---------");
        A a2 = new A();
        Sop(a2); // implicit call to toString()
        Sop("Program ends...");
    }
}
```

```
Program starts...
com.jspiders.objectclassdemo.A@1de1279
----------
com.jspiders.objectclassdemo.A@1e27869
Program ends...
```

5.* A class can override toString() method of Object class In such case the object will have overrided implementation, the original implementation will be lost.

13)

```
package com.jspiders.objectclassdemo;
class B
{
  int i;
  B(int i)
  {
    this.i = i;
  }
  public String toString()
  {
    return ""+this.i; // String format of i
  }

public class Demo3
{
  psvm(String[] args)
  {
   Sop("Program starts...");
   B b1 = new B(23);
   Sop(b1);
   Sop("----------");
   B b2 = new B(289);
   Sop(b2);
   Sop("Program ends...");
  }
}
```

O/p:

```
Program starts...
23
----------
289
Program ends...
```

equals method

14) P100

```
package com.jspiders.object class demo;

class C
{
    int i;
    C(int i)
    {
        this.i = i;
    }
}

public class Demo4
{
    psvm(string[] args)
    {
        Sop("Program starts...");
        C c1 = new C(28);
        Sop("c1 = "+c1);
        Sop("-----------");
18.     C c2 = new C(372);
19.     Sop("c2 = "+c2); 20. Sop("compare c1 and c2");
21.     Sop(c1==c2); // false
22.     Sop("-----------");
        Sop(c1.equals(c2)); // false
        Sop("Program ends...");
    }
}
```

O/P:
Program starts
c1 = com.jspiders.objectclassdemo.C@12fe278
---
c2 = com.jspiders.objectclassdemo.C@12bf18
compare c1 and c2
false

Syntax:

```
public boolean equals(Object obj)
{

}
```

1) Equals method in object class is a public method which returns boolean type. This method takes an argument type 'Object'

2) Whenever this method is invoked on an instance, the method compares the current object with passed object based on the address of object and returns either ~~false~~ true or false.

If we have to compare Object based on object fields then Equals() method has to be overrided. While overriding the current object field can be referred by using 'this' keyword, the recieved object member should be first downcasted, then its member should be referred.

P101

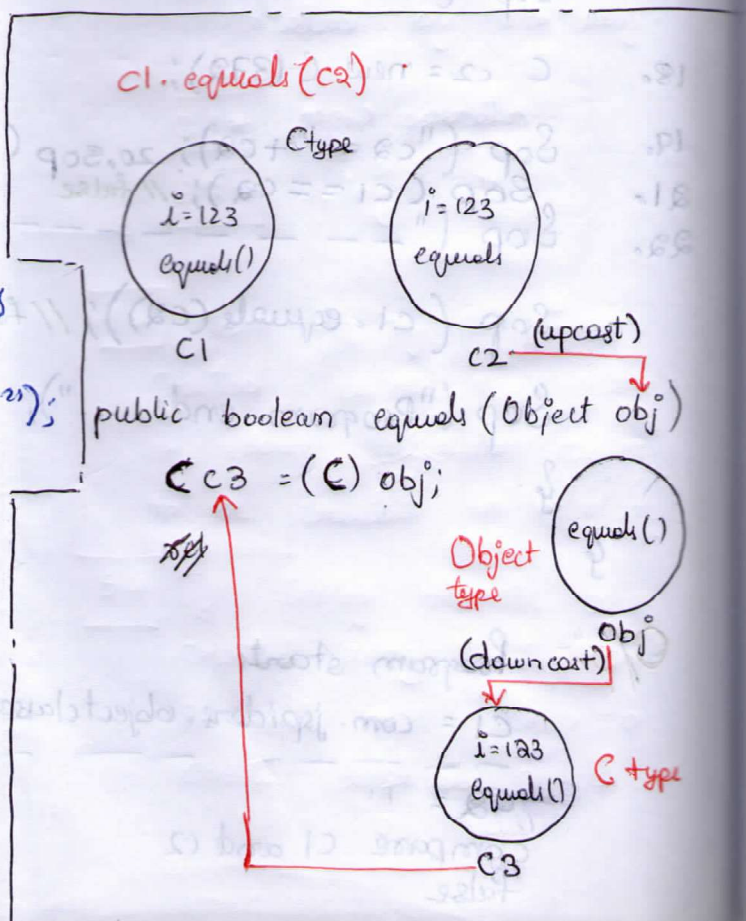
15)

```
package com.jspiders.objectclass.demo;
class C
{
    int i;
    C(int i)
    {
        this.i = i;
    }
    public boolean equals(Object obj)
    {
        C c3 = (C) obj; //downcast
        return this.i == c3.i;
    }
}
public class Demo4
{
    psvm(String[] args)
    {
        Sop("Program starts...");
        C c1 = new C(123);
        Sop("c1=" + c1);
        C c2 = new C(123);
        Sop("c2=" + c2);
        Sop("compare c1 and c2");
        Sop(c1==c2)
        Sop("-----------");
        Sop(c1.equals(c2));
        Sop("Program ends...")
    }
}
```

```
/*
    int j;

return (this.i == c3.i && this.j ==
        c3.j);
*/
```

// c2 upcasted to Object type i.e obj

O/p:- Program starts...
compare c1 and c2
false
true
Program ends...



hashcode()

15-02-2013 | Friday

16)

```
package com.jspiders.objectclassdemo;
class D
{
}
public class Demo5
{
    psvm (String[] args)
    {
        Sop ("Program starts...");
        D d1 = new D();
        Sop (d1.hashcode());
        Sop("----------");
        D d2 = new D();
        Sop (d2.hashcode());
        Sop ("Program ends...");
    }
}
```

O/p:-

```
Program starts...
30269696
----------
24052850
Program ends...
```

1*) hashcode() method of an Object class is a public method of return type integer.

Whenever this method is invoked, the method returns a hashcode value which is an integer value based on the object address.

2*) A class override an hashcode() method in such case original implementation is lost and will have overrided implementation.