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;

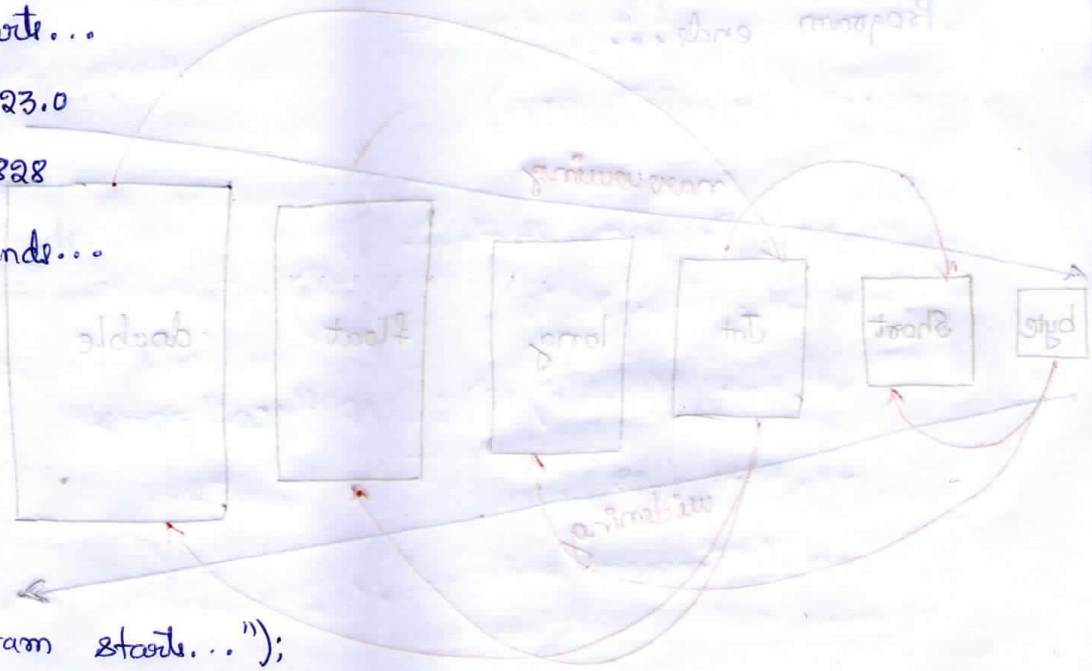
06-02-2013 | WEDNESDAY

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...



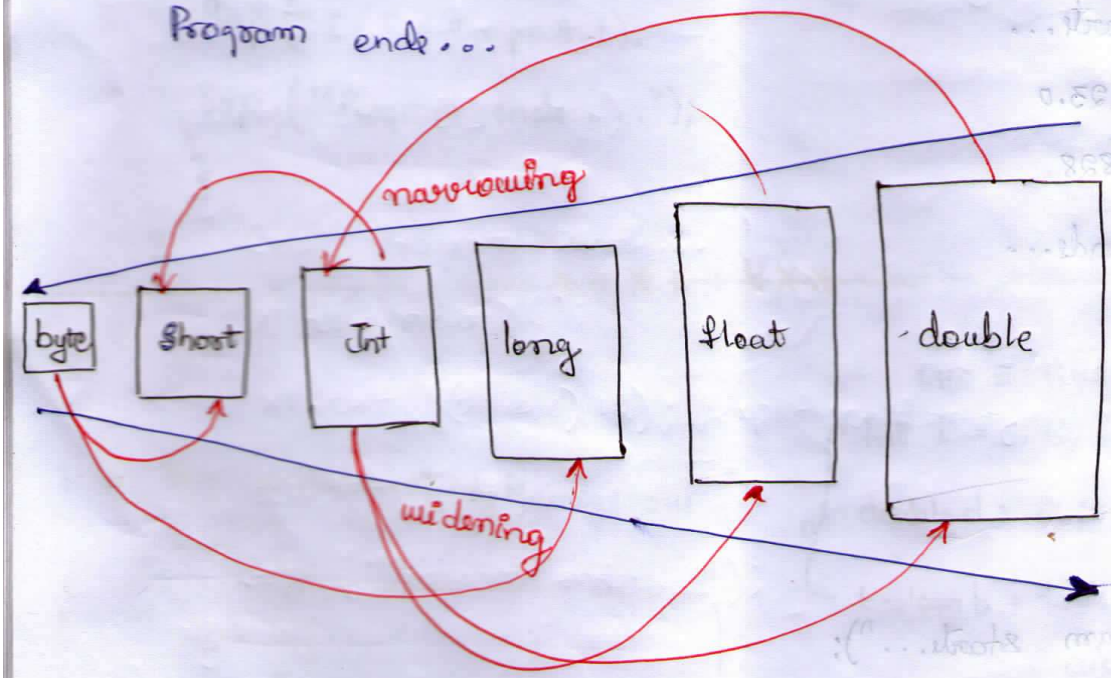
p79

```
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...

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...");
    }
}
```

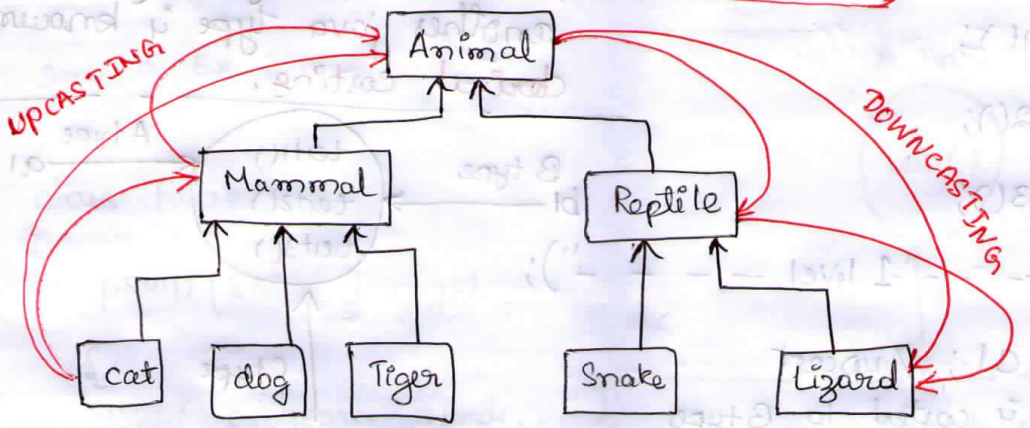
Program Start

1*) In this example class X contain overloaded test1(), one of the method taker 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 Sametype.

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

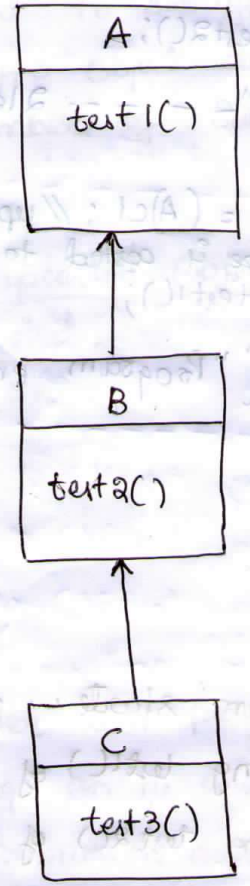
07-02-2013 THURSDAY

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...");
```

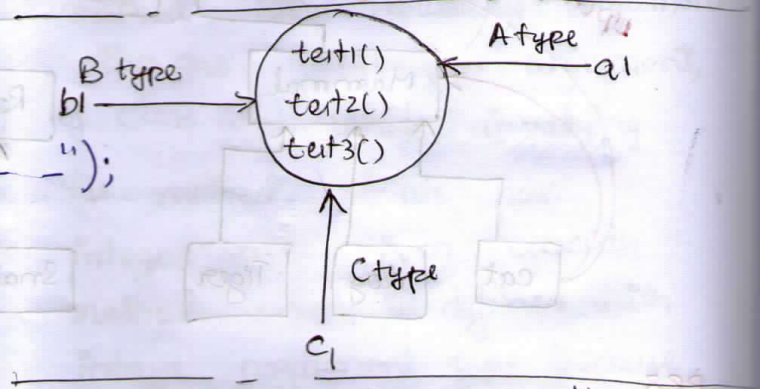


```
C c1 = new C();
c1.test1();
c1.test2();
c1.test3();
Sop("- - - - - 1 level - - - - - -");
B b1 = (B) c1; // upcast
// C type is casted to B type
b1.test1();
b2.test2();
Sop("- - - - - 2 level - - - - - -");
A a1 = (A)c1; // upcast
// C type is casted to A type.
a1.test1();
Sop("Program ends ...");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

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

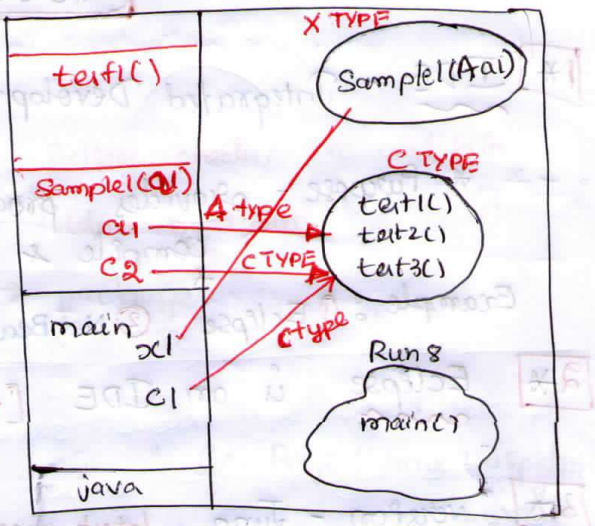
4)

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();
        c2.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...
```