Object Oriented Programming
CS F213

J. Jennifer Ranjani

email: jennifer.ranjani@pilani.bits-pilani.ac.in

Chamber: 6121 B, NAB

Consultation: Appointment by e-mail

BITS Pilani

Pilani Campus

-Arrays
-Interfaces

BITS Pilani
Pilani Campus

Answers to the Queries in previous class

1. Constructor cannot be preceded with a final keyword. Only access modifiers can be specified before a constructor
2. Class A {

```
    int I = 10; // is allowed
   A(){
    I = 20;  // I is changed and previously assigned value is of no use
    }
```

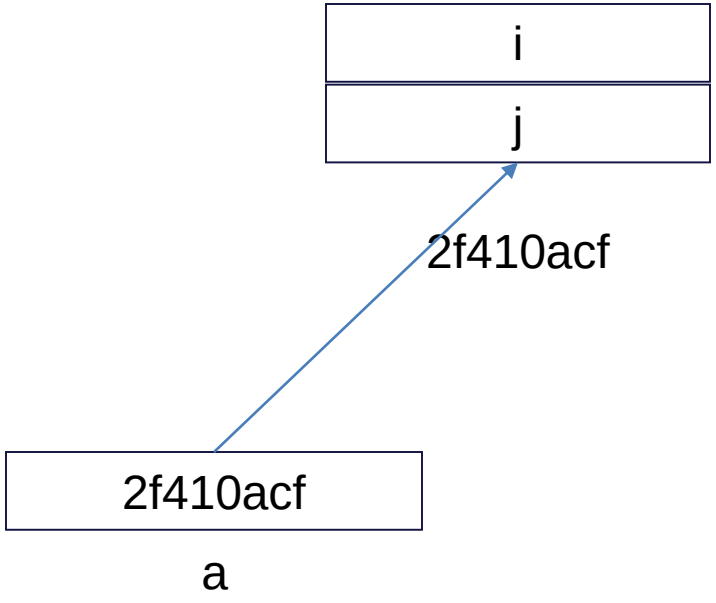
3. next(), nextInt(), nextFloat() etc ignores the leading spaces

Difference between Reference and Object

```
class zero{
int i;
float j;
}

class first{
public static void main(String args[]){

 zero a = new zero();
 System.out.println(a);
 a.i =10;
 a.j=20;
    }
}
```



Runtime Polymorphism in Multilevel Hierarchy

```
class zero{
int i=10;
float j=20;
void show() {
System.out.println(i+" "+j);}}

class first extends zero  {
int i=30;
float j=40;
void show() {
System.out.println(i+" "+j);}
}
```

```
class second extends first{
int i=50;
float j=60;
void show() {
System.out.println(i+" "+j);}
 public static void main(String
    args[]){
 zero a ;
 a = new first();
 a.show();
 first s;
s= new second();
s.show();
}
}
```

Output:
30 40.0
50 60.0

Runtime Polymorphism with Data Members

```
class zero{
int i=10;
float j=20;   }
class first extends zero  {
int i=30;
float j=40;    }
class second extends first{
int i=50;
float j=60;
 public static void main(String args[]){
 zero a ;
 a = new first();
 System.out.println(a.i+" "+a.j);
 first s;
 s= new second();
System.out.println(s.i+" "+s.j);   }  }
```

Output:
10 20.0
30 40.0

Predict the output

```
int arr1[] = {1, 2, 3};
        int arr2[] = {1, 2, 3};
        if (arr1 == arr2)
           System.out.println("Same");
        else
            System.out.println("Not same");
```

Output: Not same

Problem: == compares the array references

Solution: Arrays.equals(arr1, arr2)

Predict the output

```
int inarr1[] = {1, 2, 3};
        int inarr2[] = {1, 2, 3};
        Object[] arr1 = {inarr1};
        Object[] arr2 = {inarr2};
        if (Arrays.equals(arr1, arr2))
            System.out.println("Same");
        else
            System.out.println("Not same");
```

Output: Not same

Solution: Arrays.deepEquals(arr1, arr2)

Predict the output

```
int inarr1[] = {1, 2, 3};
    int inarr2[] = {1, 2, 3};
    Object[] arr1 = {inarr1};
    Object[] arr2 = {inarr2};
    Object[] outarr1 = {arr1};
    Object[] outarr2 = {arr2};
    if (Arrays.deepEquals(outarr1, outarr2))
        System.out.println("Same");
    else
        System.out.println("Not same");
```

Solution: Same

Final Arrays

```
class Test
{
    public static void main(String args[])
    {
        final int arr[] = {1, 2, 3, 4, 5};  // Note: arr is final
        for (int i = 0; i < arr.length; i++)
        {
            arr[i] = arr[i]*10;
            System.out.println(arr[i]);
        }
    }
}
```

Output:

10
20
30
40
50

Final Arrays

- Arrays are objects and object variables are always references in Java.

- When an object variable is declared as final, it means that the variable cannot be changed to refer to anything else.

Jagged Arrays

2D array with variable number of columns in each row.

```
        int arr[][] = new int[2][];
        arr[0] = new int[3];
        arr[1] = new int[2];
int count = 0;
 for (int i=0; i<arr.length; i++)
        for(int j=0; j<arr[i].length; j++)
                arr[i][j] = count++;
for (int i=0; i<arr.length; i++) {
        for (int j=0; j<arr[i].length; j++)
                System.out.print(arr[i][j] + " ");
                System.out.println();   }
```

Printing a 2D Array using toString


```
int inarr1[] = {1, 2, 3};
    int inarr2[] = {4, 5, 6};

    Object arr[] = {inarr1,inarr2};
    System.out.println(Arrays.deepToString(arr));

    int a[][]= {{1,2,3},{4,5}};
    System.out.println(Arrays.deepToString(a));

    System.out.println(a[1].length);
```

Output:

```
[[1, 2, 3], [4, 5, 6]]
[[1, 2, 3], [4, 5]]
2
```

More on Final Objects


```
class Test
{
    int p = 20;
    public static void main(String
     args[])
    {
       final Test t = new Test();
       t.p = 30;
       System.out.println(t.p);
    }
}
```

Output: 30

```
class Test
{
    int p = 20;
    public static void main(String
     args[])
    {
       final Test t1 = new Test();
       Test t2 = new Test();
       t1 = t2;
       System.out.println(t1.p);
    }
}
```

Compiler Error

innovate achieve lead

BITS Pilani

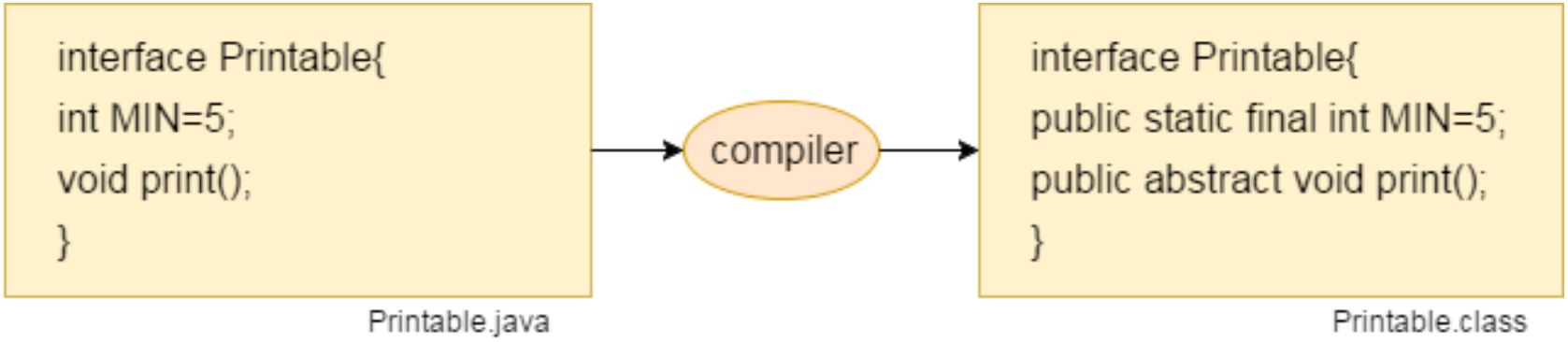
Pilani Campus

Interfaces

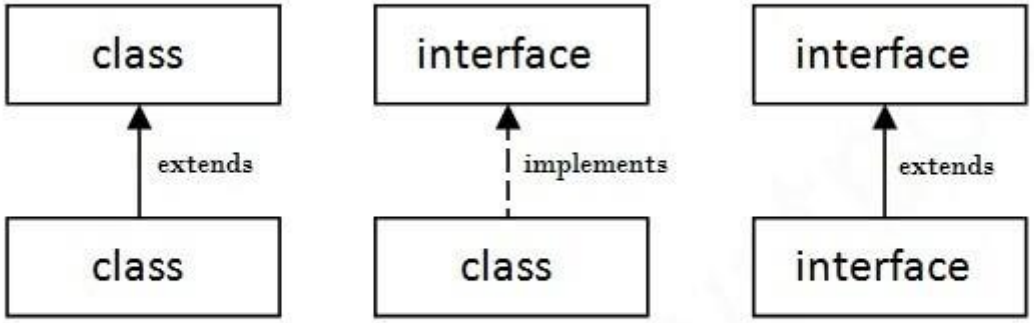
Interface

- Interface is a blueprint of a class containing static constants and abstract methods. It cannot have a method body.

- It is a mechanism to achieve abstraction.



Relationship between Classes and Interfaces



Interfaces - Example

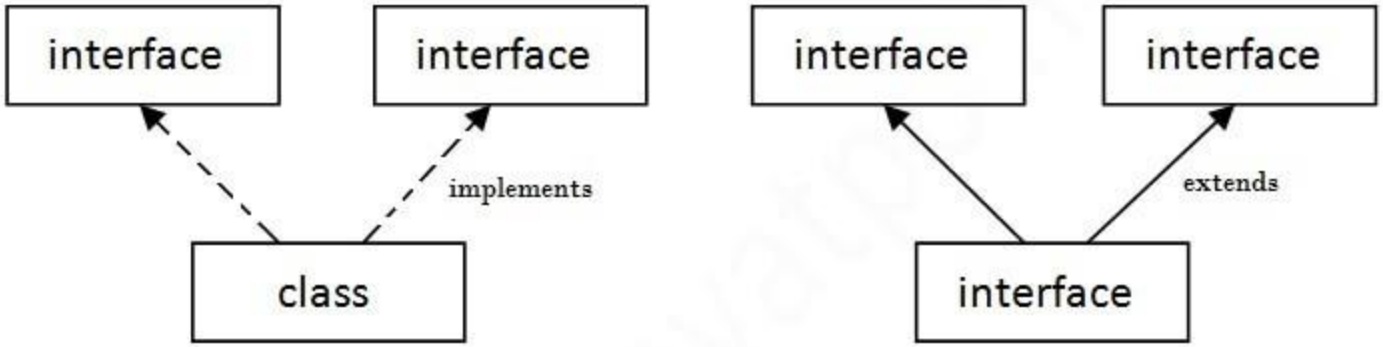
innovate achieve lead

```
Interface Bank {
 void deductFee();
 void withdraw(float amount);
 void deductFee();}

class BankAccount implements Bank{
.
.
public void deductFee();{}
}

class CheckingAccount extends BankAccount implements Bank
{
.
.
.
}
```

Multiple Inheritance in Interface



Multiple Inheritance in Java

Why is Multiple Inheritance not a problem in Interface?

innovate achieve lead

```
interface Printable{
void print();
void show();   }
interface Showable{
void show();
void print();  }

class trial implements
     Printable,Showable {
public void show()  {
System.out.println("Within Show");}

public void print() {
System.out.println("Within Print");}
}
```

```
public class test {
public static void main(String[]
     args) {
trial t = new trial();
t.print();
t.show();
}
}
```

Default Methods in Interface (defender or virtual extension)

- Before Java 8, interfaces could have only abstract methods. Implementation is provided in a separate class

- If a new method is to be added in an interface, implementation code has to be provided in all the classes implementing the interface.

- To overcome this, default methods are introduced which allow the interfaces to have methods with implementation without affecting the classes.

Default Methods

```
interface Printable{
void print();
default void show()
{
System.out.println("Within Show");
}
}

class trial implements Printable {

public void print()
{
System.out.println("Within Print");
}
}
```

```
public class test {
public static void main(String[]
    args) {
trial t = new trial();
t.print();
t.show();
}

}
```

Default Methods & Multiple Inheritance

innovate achieve lead

```
interface Printable{
void print();
default void show()
{
System.out.println("Within
    Printable Show");
}
}
interface Showable{
default void show()
{
System.out.println("Within
    Showable Show");
}
void print();
}
```

```
class trial implements Printable,Showable{
public void show()  {
Printable.super.show();
Showable.super.show(); }

public void print() {
System.out.println("Within Print"); }}

public class test {
public static void main(String[] args) {
trial t = new trial();
t.print();
t.show();
}
}
```