Object Oriented Programming
CS F213
Amit Dua

BITS Pilani
Pilani Campus

Slides Taken from the slides prepared by Dr. Jennifer

innovate achieve lead

BITS Pilani

Pilani Campus

‘Final’ Keyword

Java Final Keyword

- Makes variable a constant
- Prevents Method Overriding
- Prevents Inheritance

Blank or uninitialized final variable

- A final variable that is not initialized at the time of declaration is known as blank final variable.

- It can be used when variable is initialized at the time of object creation and should not be changed after that.
  - Eg. Pan card

- It can be initialized only once (preferably within a constructor).

Final blank variable

innovate achieve lead

Example 1:

```
class first{

 public static void main(String
     args[]){
    final int i;
    i=10;

       System.out.println("s1: "+i);
       i=20;  // Error

   }
}
```

Example 2:

```
class first{
final int i;
i=10 // Error
first(){
i=10;
}

 public static void main(String
     args[]){

       System.out.println("s1: "+new
    first().i);
   }
}
```

Static Blank Final Variable

- A static final variable that is not initialized at the time of declaration is known as static blank final variable. It can be initialized only in static block.

```
class A{
  static final int data;//static blank final variable
  static{ data=50;}
  public static void main(String args[]){
    System.out.println(A.data);
 }
}
```

Questions?

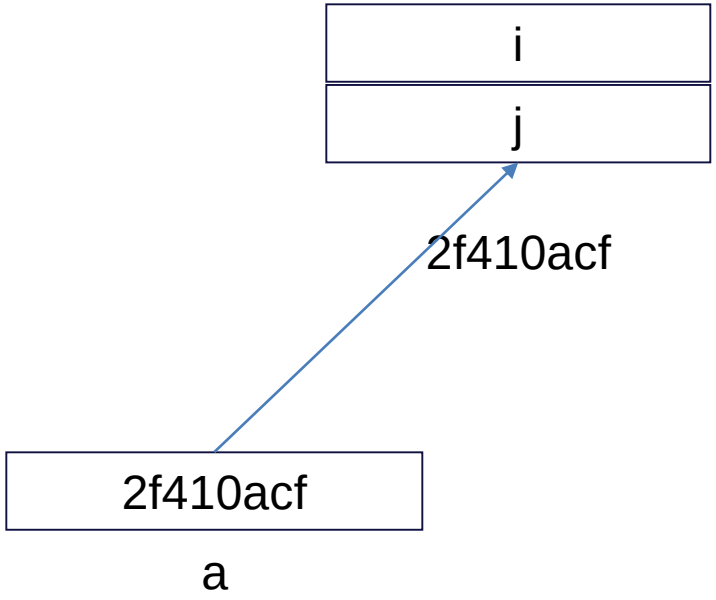
- Is final method inherited?
  - YES. But it cannot be overridden
- Can we declare a constructor final?
  - NO. Constructor is not inherited

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;
    }
}
```



innovate achieve lead

BITS Pilani

Pilani Campus

Run Time Polymorphism

Dynamic Method Dispatch

- Method overriding is one of the ways in which Java supports Runtime Polymorphism.
- Dynamic method dispatch is the mechanism by which a call to an overridden method is resolved at run time, rather than compile time.
- An overridden method is called through the reference variable of a superclass.
- The determination of the method to be called is based on the object being referred to by the reference variable.
- **Upcasting**: The reference variable of Parent class refers to the object of Child class.

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

Static vs. Dynamic Binding (Early vs. Late Binding)

- Static binding happens at compile-time while dynamic binding happens at runtime.
- Binding of private, static and final methods always happen at compile time since these methods cannot be overridden.
- When the method overriding is actually happening and the reference of parent type is assigned to the object of child class type then such binding is resolved during runtime.
- The binding of overloaded methods is static and the binding of overridden methods is dynamic.

Bank - Example

innovate achieve lead

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

Scanner sr = new Scanner(System.in);

System.out.println("Enter 1 for new customers (< 1 year) and 0 for others");
int yr = sr.nextInt();

BankAccount ba;

if (yr==1)
ba = new BankAccount(111,"Ankit",5000);
else
ba = new CheckingAccount(111,"Ankit",5000);
```

Bank - Example


```
System.out.println("Initial: "+ba.getBalance());

ba.deposit(1000);
ba.withdraw(2000);
ba.deposit(6000);
System.out.println("After three Transactions: " + ba.getBalance());

ba.deductFee();        //ERROR

System.out.println("After fee Deduction: " + ba.getBalance());
sr.close();

}}
```

Solution 1

- Create an empty method in the Bank Account class

```
void deductFee()
{
}
```

- Meaningless, Isn’t it?

Solution 2 – Abstract Class

```
abstract class BankAccount{
private int acc;
private String name;
private float amount;

BankAccount(int acc,String name,float amt)
{
this.acc = acc;
this.name = name;
this.amount = amt;   }

void setAcc(int acc)   {
this.acc = acc;    }

void setName(String name)  {
this.name = name;   }

float getBalance(){
return amount;}

void deposit(float amount)   {
this.amount = this.amount+amount; }

void withdraw(float amount)  {
if (this.amount < amount)
System.out.println("Insufficient
     Funds. Withdrawal Failed");
else
this.amount=this.amount-amount;  }

abstract void deductFee();
}
```

Abstract class

- an instance of an abstract class cannot be created, we can have references of abstract class type
- an abstract class can contain constructors
- we can have an abstract class without any abstract method
- Abstract classes can also have final methods (methods that cannot be overridden)

Example abstract class

```
abstract class Base
{
    final void fun()
{System.out.println("Der
   ived fun() called"); }
}
class Derived extends
   Base { }
```

```
class Main {
    public static void
    main(String args[])
    {
        Base b = new
    Derived();
        b.fun();
    }
}
```

Questions

- Is it possible to create abstract and final class in Java?
- Is it possible to have an abstract method in a class?
- Is it possible to have an abstract method in a final class?
- Is it possible to inherit from multiple abstract classes in Java?

innovate achieve lead

BITS Pilani

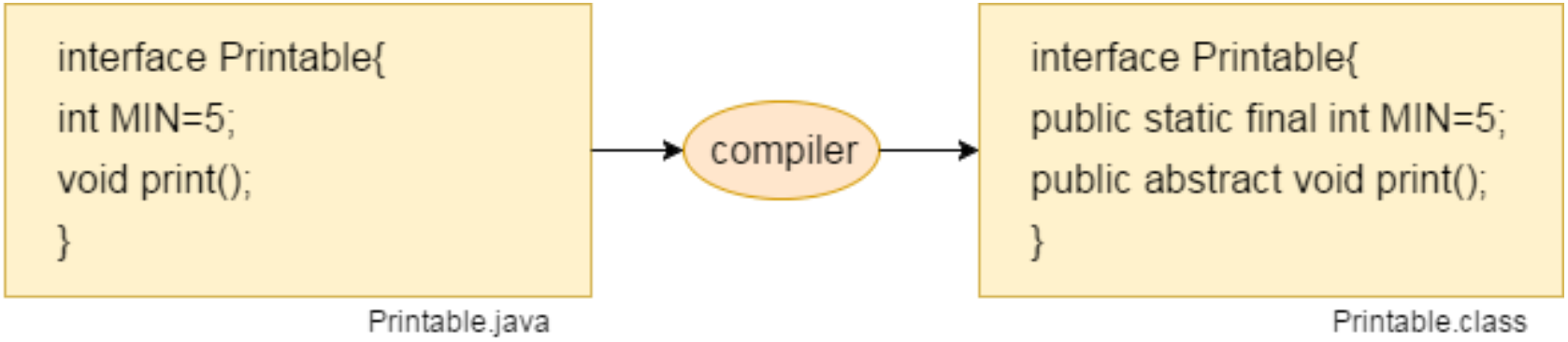
Pilani Campus

Interfaces

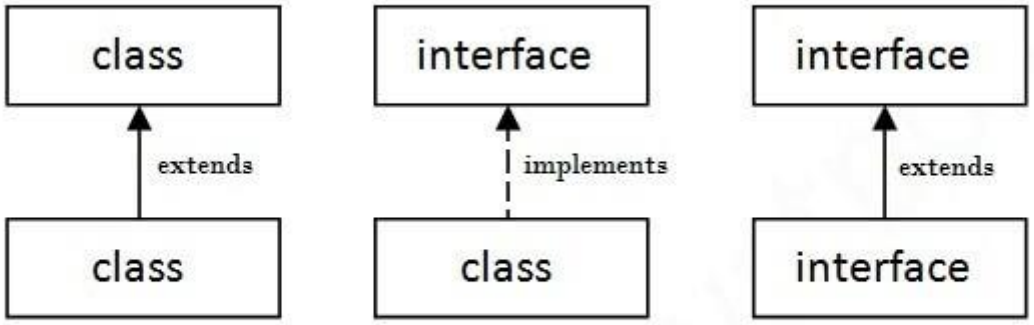
Interface

- 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

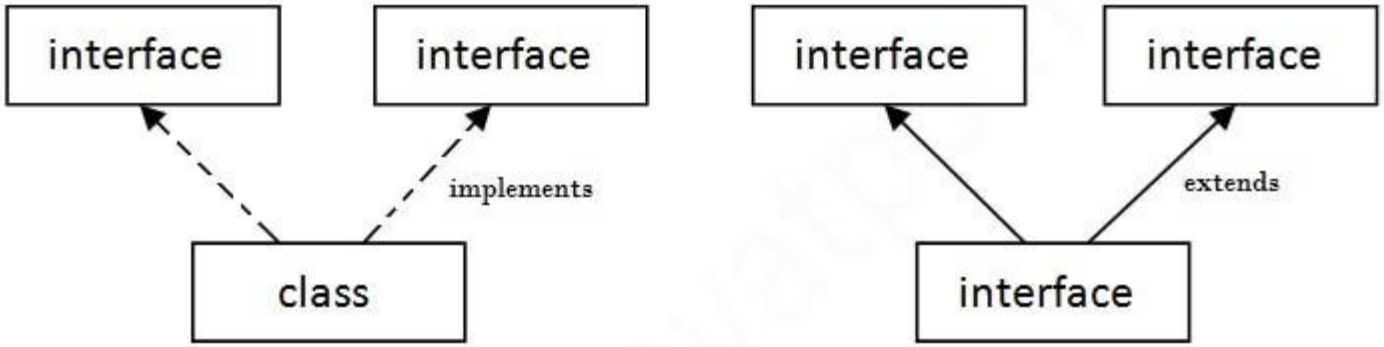
innovate achieve lead

```
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?


```
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();
}
}
```