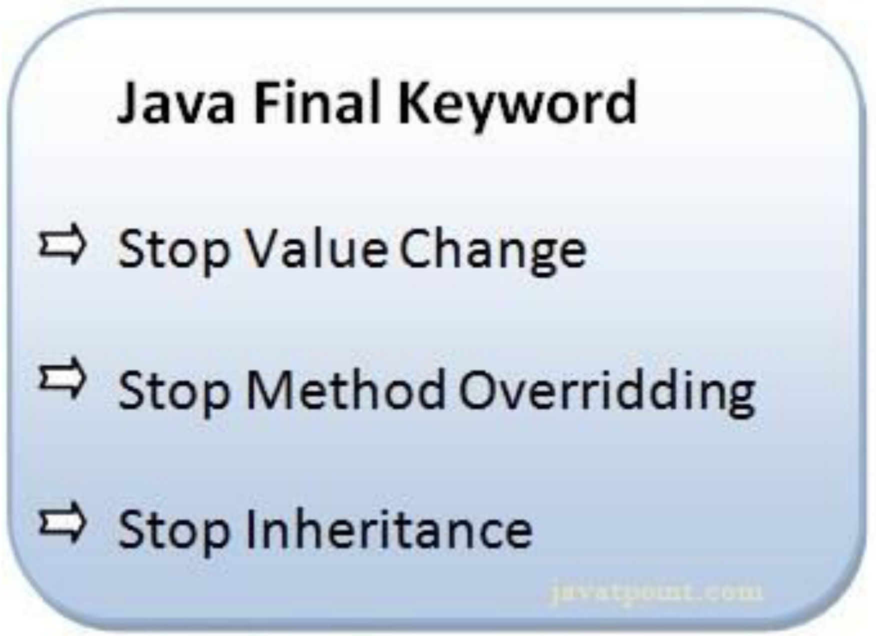
Programming Language II
CSE-215

Prof. Dr. Mohammad Abu Yousuf
yousuf@juniv.edu

Final Keyword in java

- The **final keyword** in java is used to restrict the user. The java final keyword can be used in many context. Final can be:
  - variable
  - method
  - Class
- The final keyword can be applied with the variables, a final variable that have no value it is called blank final variable or uninitialized final variable. It can be initialized in the constructor only. The blank final variable can be static also which will be initialized in the static block only.

Final Keyword in java



Java Final Keyword

⇨ Stop Value Change

⇨ Stop Method Overridding

⇨ Stop Inheritance

Java final variable

- If you make any variable as final, you cannot change the value of final variable(It will be constant).

```
class Bike9{
 final int speedlimit=90;//final variab
le
 void run(){
  speedlimit=400;
 }
 public static void main(String args[])
{
 Bike9 obj=new  Bike9();
 obj.run();
 }
}//end of class
```

There is a final variable speedlimit, we are going to change the value of this variable, but It can't be changed because final variable once assigned a value can never be changed.

```
Output: Compile Time Error
```

Java final method

- If you make any method as final, you cannot override it.

```
class Bike{
 final void run(){System.out.println("running");}
}
    class Honda extends Bike{
  void run(){
     System.out.println("running safely with 100kmph");
  }
  public static void main(String args[]){
     Honda honda= new Honda();
     honda.run();
  }
}
```

```
Output: Compile Time Error
```

Java final class

- If you make any class as final, you cannot extend it.

```
final class Bike{}

class Honda1 extends Bike{
  void run(){
      System.out.println("running safely with 100kmph");
  }
  public static void main(String args[]){
  Honda1 honda= new Honda();
  honda.run();
  }
}
```

```
Output: Compile Time Error
```

Is final method inherited?

- Yes, final method is inherited but you cannot override it. For Example:

```
class Bike{
  final void run(){System.out.println("running...");}
}
class Honda2 extends Bike{
   public static void main(String args[]){
       new Honda2().run();
   }
}
```

```
Output: running...
```

What is blank or uninitialized final variable?

- A final variable that is not initialized at the time of declaration is known as blank final variable.
- If you want to create a variable that is initialized at the time of creating object and once initialized may not be changed, it is useful.

```
class Student{
int id;
String name;
final String PAN_CARD_NUMBER;
...
}
```

Can we initialize blank final variable?

- Yes, It can be initialized but only in constructor. For example:

```
class Bike10{
  final int speedlimit;//blank final variable
  Bike10(){
  speedlimit=70;
  System.out.println(speedlimit);
  }
  public static void main(String args[]){
     new Bike10();
 }
}
```

```
Output: 70
```

What is final parameter?

- If you declare any parameter as final, you cannot change the value of it.

```
class Bike11{
  int cube(final int n){
   n=n+2;//can't be changed as n is final
   n*n*n;
  }
  public static void main(String args[]){
    Bike11 b=new Bike11();
    b.cube(5);
 }
}
```

Output: Compile Time Error

Can we declare a constructor final?

- No, because constructor is never inherited.

Polymorphism in Java

Polymorphism

- **Polymorphism in java** is a concept by which we can perform a *single action by different ways*. Polymorphism is derived from 2 greek words: poly and morphs. The word "poly" means many and "morphs" means forms. So polymorphism means many forms.
- There are two types of polymorphism in java: **compile time polymorphism and runtime polymorphism.** We can perform polymorphism in java by method overloading and method overriding.
- **Method overloading** is compile time polymorphism.
- **Method overriding** is run time polymorphism.

Runtime Polymorphism in Java

- Let's first understand the upcasting before Runtime Polymorphism.

Upcasting:

- When reference variable of Parent class refers to the object of Child class, it is known as upcasting. For example:

```
class A{}
class B extends A{}
A a=new B();//upcasting
```

Example of Java Runtime Polymorphism

```
class Bike{
 void run(){System.out.println("running");}
}
class Splender extends Bike{
 void run(){
      System.out.println("running safely with 60km");
 }
 public static void main(String args[]){
   Bike b = new Splender();//upcasting
   b.run();
 }
}
```

Output:

```
running safely with 60km.
```

we are creating two classes Bike and Splendar. Splendar class extends Bike class and overrides its run() method. We are calling the run method by the reference variable of Parent class. Since it refers to the subclass object and subclass method overrides the Parent class method, subclass method is invoked at runtime.

Another Example of Java Runtime Polymorphism

```
class Shape{
void draw(){System.out.println("drawing...");}
}
class Rectangle extends Shape{
void draw(){System.out.println("drawing rectangle...");}
}
class Circle extends Shape{
void draw(){System.out.println("drawing circle...");}
}
class Triangle extends Shape{
void draw(){System.out.println("drawing triangle...");}
}
class TestPolymorphism2{
public static void main(String args[]){
Shape s;
s=new Rectangle();
s.draw();
s=new Circle();
s.draw();
s=new Triangle();
s.draw();
}  }
```

Output:

```
drawing rectangle...
drawing circle...
drawing triangle...
```

Java Runtime Polymorphism with Data Member

Method is overridden not the data members, so runtime polymorphism can't be achieved by data members.

In the example (next slide), both the classes have a data member speedlimit, we are accessing the data member by the reference variable of Parent class which refers to the subclass object. Since we are accessing the data member which is not overridden, hence it will access the data member of Parent class always.

Java Runtime Polymorphism with Data Member

```
class Bike{
 int speedlimit=90;
}
class Honda3 extends Bike{
 int speedlimit=150;

 public static void main(String args[]){
  Bike obj=new Honda3();
  System.out.println(obj.speedlimit);//90
}
```

```
Output: 90
```

Rule: Runtime polymorphism can't be achieved by data members.

Java Runtime Polymorphism with Multilevel Inheritance

```
class Animal{
void eat(){System.out.println("eating");}
}
class Dog extends Animal{
void eat(){System.out.println("eating fruits");}
}
class BabyDog extends Dog{
void eat(){System.out.println("drinking milk");}
public static void main(String args[]){
Animal a1,a2,a3;
a1=new Animal();
a2=new Dog();
a3=new BabyDog();
a1.eat();
a2.eat();
a3.eat();
}
}
```

Output:

```
eating
eating fruits
drinking Milk
```

Try for Output

```
class Animal{
void eat(){System.out.println("animal is eating...");}
}
class Dog extends Animal{
void eat(){System.out.println(“Dog is eating...");}
}
class BabyDog1 extends Dog{
public static void main(String args[]){
Animal a=new BabyDog1();
a.eat();
}}
```

Output:
Dog is eating

Since, BabyDog1 is not overriding the eat() method, so eat() method of Dog class is invoked.

Thank you