

Object Oriented Programming

Prepared by Dr. Yasser Abdelhamid

IT223

Object Oriented Programming

Lecture 05



OUTLINE

- Inheritance.
- Access modifiers and inheritance
- Method overriding
- Method overloading.
- Calling superclass methods
- Programming challenge

INHERITANCE

INHERITANCE

- ❖ In Java, it is possible to inherit attributes and methods **from one class to another**.
- ❖ We can create new classes from existing classes.
- ❖ Inheritance supports **reusability**.

INHERITANCE (CONT.)

Class A



Class B

```
public class A {
    ...
}
```

```
public class B extends A {
    ...
}
```

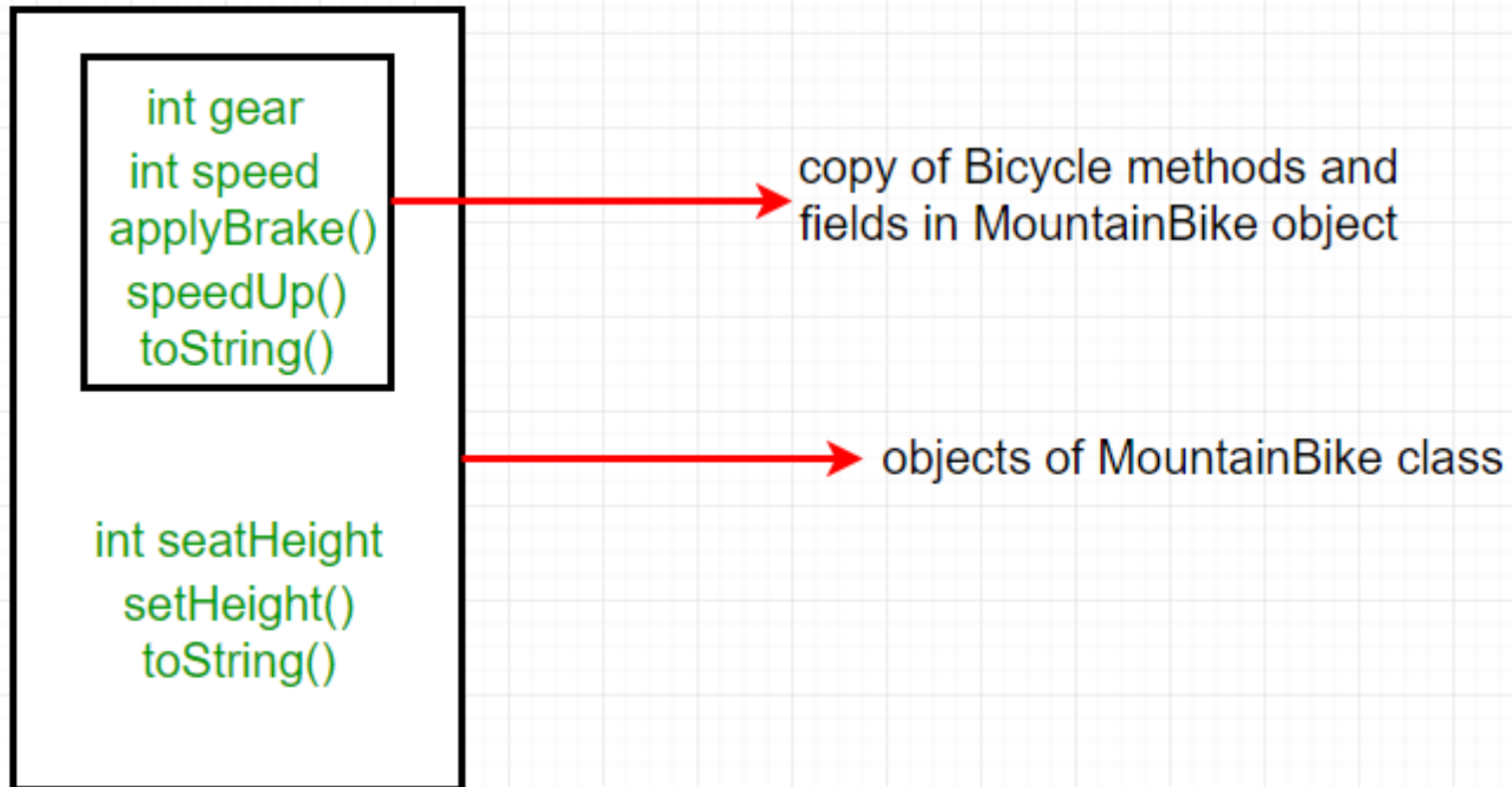
Superclass, Parent, Base class.

Subclass, Derived, Child, Extended class.

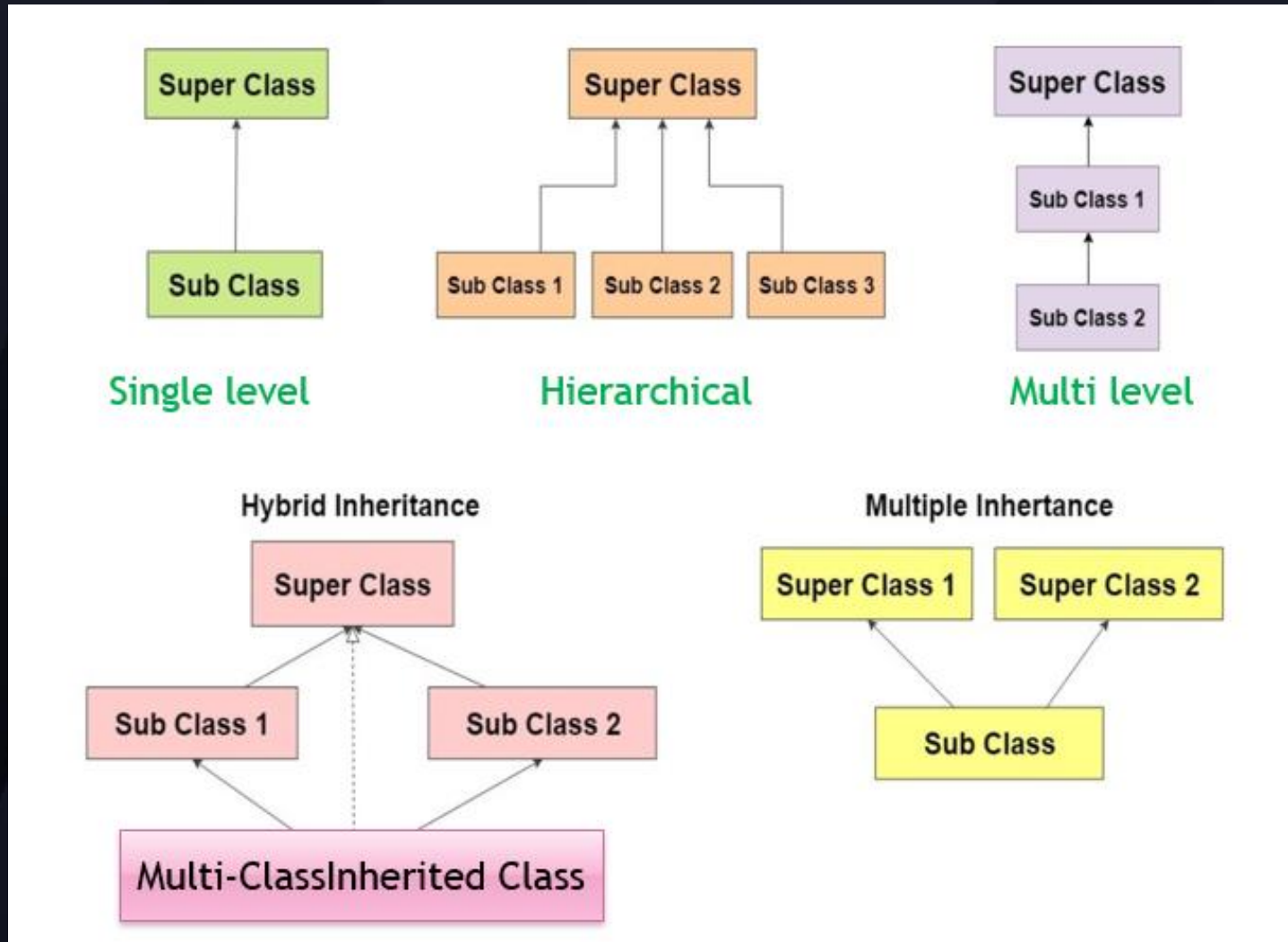
INHERITANCE

- ❖ When we create a class based on an existing class, we can reuse methods and fields of the **parent class**.
- ❖ We can add new methods and fields in our new **child class**.

INHERITANCE (CONT.)



INHERITANCE



EXAMPLE: SINGLE LEVEL INHERITANCE

```
class Employee{  
    double basicSalary =40000.0;  
  
}  
  
class Programmer extends Employee{  
    double bonus=10000;  
  
    public static void main(String[] args){  
        Programmer p=new Programmer();  
        System.out.println("Programmer basic salary is:\t"+p.basicSalary);  
        System.out.println("Bonus of Programmer is:"+p.bonus);  
    }  
}
```

MULTI LEVEL INHERITANCE

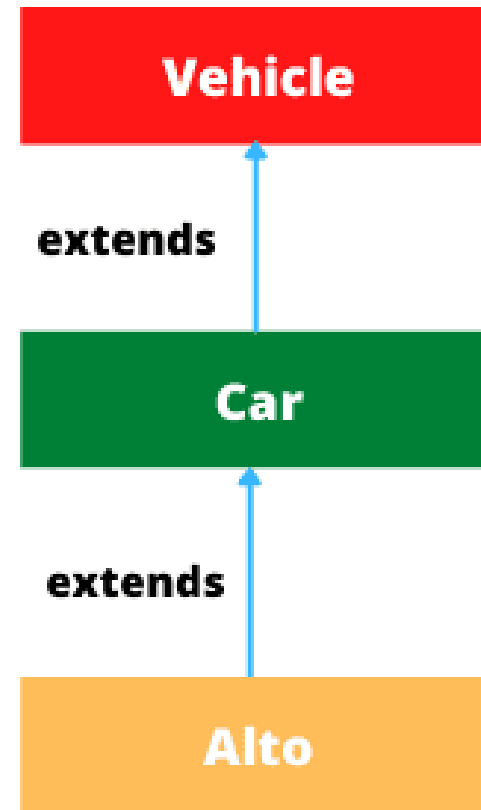
```
public class Vehicle {
    .....
    .....
}

public class Car extends Vehicle {
    .....
    .....
}

public class Alto extends Car {
    .....
    .....
}
```

IS-A Relationship

IS-A Relationship



EXAMPLE: MULTI LEVEL INHERITANCE

```
class Animal{
void eat() {System.out.println("eating...");}
}
class Dog extends Animal{
void bark() {System.out.println("barking...");}
}
class BabyDog extends Dog{
void breastfed() {System.out.println("breastfed...");}
}
class TestInheritance2{
    public static void main(String args[]){
        BabyDog d=new BabyDog();
        d.breastfed();
        d.bark();
        d.eat();
    }
}
```

EXAMPLE: HIERARCHICAL INHERITANCE

```
class Animal{
    void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
    void bark(){System.out.println("barking...");}
}
class Cat extends Animal{
    void meow(){System.out.println("meowing...");}
}

class TestInheritance3{
    public static void main(String args[]){
        Cat c=new Cat();
        c.meow();
        c.eat();
    }
}
```

THE OBJECT CLASS

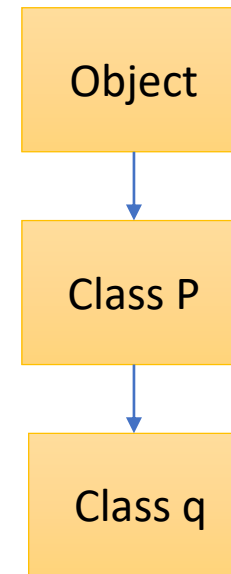
- ❖ Object Class is the Default Superclass.
- ❖ If a class does not specify a superclass using the keyword `extends` in its class declaration, it inherits from the `java.lang.Object` class.

- ❖ We usually declare a class like this

```
public class P {  
    // Code for class P goes here  
}
```

- ❖ We can declare a class explicitly like this

```
public class P extends Object {  
    // Code for class P goes here  
}
```



ACCESS MODIFIERS AND INHERITANCE

ACCESS MODIFIERS AND INHERITANCE

❖ A subclass inherits **non-private** members of its superclass.

Public ✓

Default ✓

Private ✗

Protected ✓

EXAMPLE: PRIVATE ACCESS MODIFIER

```
class TestInheritance {  
    private int x;  
    int y;  
  
    public TestInheritance(){  
        x=10;  
        y=9;  
    }  
}  
  
class Test extends TestInheritance{  
    public static void main(String[] args) {  
        Test a = new Test();  
        System.out.println(a.y);  
        System.out.println(a.x); \\ won't work  
    }  
}
```

PROTECTED MODIFIER

❖ There are two cases:

- Protected member is accessible in the package in which it is declared.
- A protected class member is always accessible inside the body of a subclass whether the subclass is in the same package as the class or in a different package.

EXAMPLE: PROTECTED MODIFIER

```
package myTestClasses;
public class TestInheritance {
    protected int p;
    public TestInheritance(){
        p=88;
    }
}
class A {
    public static void main(String[] args) {
        TestInheritance a = new TestInheritance();
        a.p = 77; // protected members are accessible in the same package
    }
}
```

ACCESSING PROTECTED MEMBER FROM OTHER PACKAGE

```
package package1;
import myTestClasses.TestInheritance;
class Test extends TestInheritance {
    public static void main(String[] args) {
        Test t = new Test();
        t.p = 777;
        TestInheritance t2 = new TestInheritance();
        t2.p = 888;
    }
}
```

METHOD OVERRIDING

METHOD OVERRIDING

- ❖ Child class can declare methods with the same name of a method declared in the parent class.
- ❖ This is known as **method overriding** in Java.
- ❖ This technique is usually used to develop **specific implementation** of the method that has been declared by its parent class.

EXAMPLE: METHOD OVERRIDING

```
class Vehicle{  
    //defining a method  
    void run(){System.out.println("Vehicle is running");}  
}  
//Creating a child class  
class Bike2 extends Vehicle{  
    //defining the same method as in the parent class  
    void run(){System.out.println("Bike is running safely");}  
  
    public static void main(String args[]){  
        Bike2 obj = new Bike2();//creating object  
        obj.run();//calling method  
    }  
}
```

METHOD OVERRIDING RULES

1. The method name of the child class must be the **same name** as the method of the parent class.
2. The two methods (of the child class and parent class) must have the **same parameters**.
3. method must **not be static**.
4. The method declared in child class **must not be more restrictive** (must be less restrictive).

QUESTION?

Can we override java main method?

No, because it is static

QUESTION?

Would this code run?

```
class ParentClass{  
    void msg(){System.out.println("Overriden method");}  
}  
  
class ChildClass extends ParentClass{  
    protected void msg(){System.out.println("Overriding method");}  
  
    public static void main(String args[]){  
        ChildClass obj=new ChildClass();  
        obj.msg();  
    }  
}
```

Yes , because
Overriding method is
less restrictive.

VISIBILITY ORDER OF ACCESS MODIFIERS FROM LEAST TO MOST

- ❖ **** private
- ❖ *** default
- ❖ ** protected
- ❖ * public

Memorize this sequence
PRI DEF PRO PUB

METHOD OVERLOADING

METHOD OVERLOADING

- ❖ Declaring multiple methods with the **same name and different parameters**.
- ❖ Method overloading increases the **readability** of the program.
- ❖ Overloading can be implemented by:
 - Changing number of parameters.
 - Changing the data type of parameters.

QUESTION?

Method Overloading is possible by **Only** changing the return data type of the method. (True/False)

Justify your answer

No, a conflict occurs.

EXAMPLE: METHOD OVERLOADING

```
class Operator{
    static int add(int a,int b){return a+b;}
    static int add(int a,int b,int c){return a+b+c;}
    static double add(double a,double b){return a+b;}
}

class TestOverloading1{
    public static void main(String[] args){
        System.out.println(Operator.add(1,2));
        System.out.println(Operator.add(1,2,3));
        System.out.println(Operator.add(1d,2d));
    }
}
```

QUESTION?

```
public class MainOverloading {  
    // This version must be included  
    public static void main(String[] args) {  
        System.out.println("all ok");  
        main();  
        main("Hello");  
    }  
    public static void main() {  
        System.out.println("No arguments");  
    }  
    public static void main(String o) {  
        System.out.println(o);  
    }  
}
```

Can we overload java
main() method?



YEAH!

CALLING SUPERCLASS METHODS

CALLING SUPERCLASS METHODS

- ❖ Creating an instance of a subclass, implicitly creates a superclass instance.
- ❖ The superclass instance is referenced by **super** keyword.
- ❖ **super** keyword can be used as follows:
 - Referring to immediate parent class **instance variable**.
 - Invoking immediate parent **class method**.
 - Invoking immediate parent class **constructor**.

CALLING IMMEDIATE PARENT CLASS INSTANCE VARIABLE

```
class Animal{
    String color="white";
}
class Dog extends Animal{
    String color="black";
    void printColor(){
        System.out.println(color);
        System.out.println(super.color);
    }
}
class TestSuper1{
    public static void main(String args[]){
        Dog d=new Dog();
        d.printColor();
    }
}
```

CALLING PARENT CLASS METHODS

```
class Animal{  
    void eat(){  
        System.out.println("eating...");  
    }  
}  
class Dog extends Animal{  
    void eat(){  
        System.out.println("eating bread..."); }  
    void bark(){  
        System.out.println("barking..."); }  
    void work(){  
        super.eat();  
        bark();  
    }  
}
```

```
class TestSuper2{  
    public static void main(String args[]){  
        Dog d=new Dog();  
        d.work();  
    }  
}
```

CALLING PARENT CLASS CONSTRUCTOR

```
class Animal{
    Animal(){
        System.out.println("animal is created");
    }
}
class Dog extends Animal{
    Dog(){
        super(); // What if I removed this line?
        System.out.println("dog is created");
    }
}
class TestSuper3{
    public static void main(String args[]){
        Dog d=new Dog();
    }
}
```

Nothing happens because it calls the superclass constructor by default.

PROGRAMMING CHALLENGE

PROGRAMMING CHALLENGE

Electricity invoices in Egypt are calculated in slices as follows:

1. First 50 KWatts cost L.E. 0.80 /KW
2. Next 150 KWatts cost L.E. 0.90 /KW
3. Next 150 KWatts cost L.E. 1.00 /KW
4. More than 350 KWatts cost L.E. 1.20 /KW

Develop a Java application to calculate the electricity invoice for customers.

The application should print the invoice which include the following information:

Customer name, Customer Address, Amount of KWatts consumed, and Amount of the invoice in L.E. calculated as specified above.

The application will ask the user for his name and address and consumed KWatts.

Hint. You have to create classes for Slices, Customers, Invoices, In addition to the main application class, and any other classes that you may need in the application.

Use OOP principles like encapsulation, inheritance as possible.

SUMMARY

- Inheritance.
- Access modifiers and inheritance
- Method overriding
- Method overloading.
- Calling superclass methods
- Programming challenge



Thank You

www.midocean.ae

