JAVA @17

Inheritance

Objective

After completing this session, you will be able to understand,

- Define Inheritance
- Super class and Sub class
- Super constructor
- Method overriding
- Run Time Polymorphism

Lets look at some model of phones that evolved in the past two decades. All the phones can be used to call friends(or anyone) and receive calls from them.
So how do they differ?

What is inheritance?

Basic Phone – People can talk

Smart Phone People can,
- **Talk**
- **Send SMS**
- **Browse the Web**
- **Take Photos/Videos**

Cordless Phone People can,
- **Talk**

Mobile Phone People can,
- **Talk**
- **Send SMS**

Each of them have INHERITED the common functionality *talking* from the ancestral basic phone and have some add-on features (SMS, Web browsing) of their own.
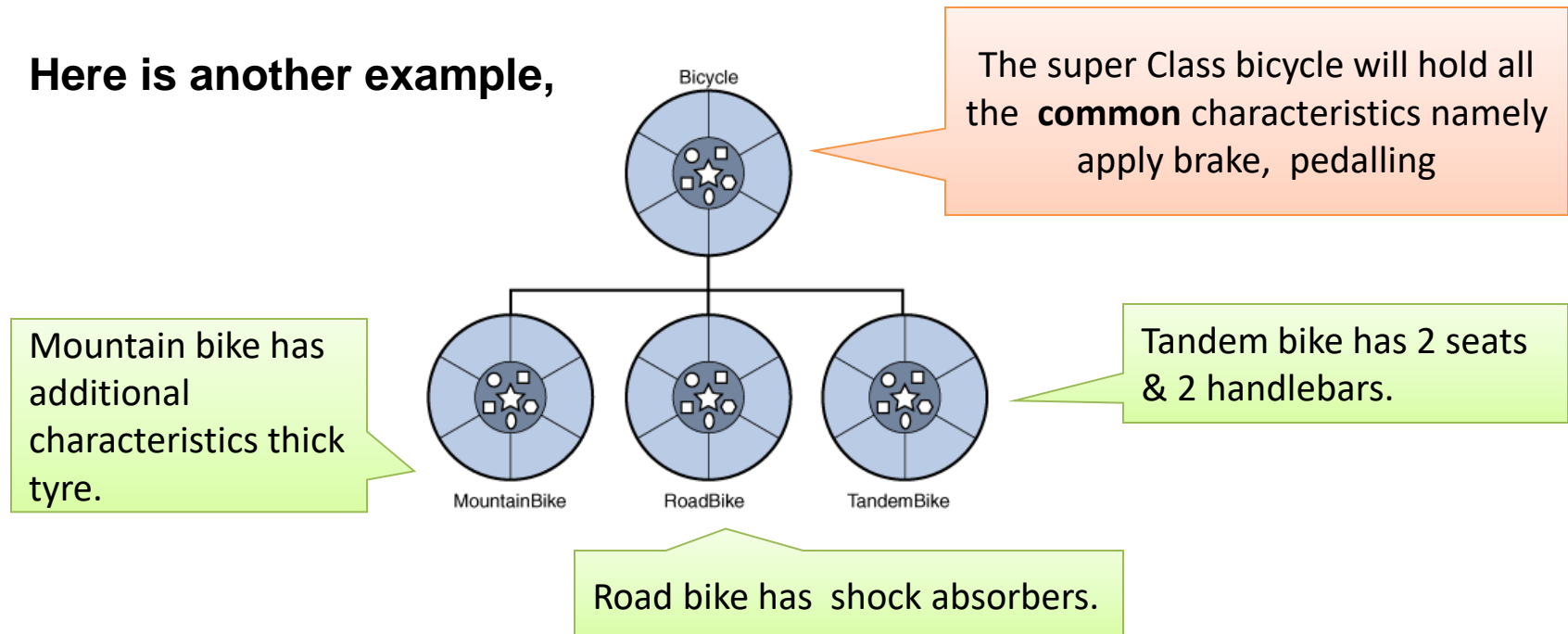This is *Inheritance*.

Inheritance

What is inheritance?

Inheritance is the concept of a **child** class (*sub class*) automatically **inheriting** the variables and methods defined in a **parent** class (*super class*).

It is one of the primary features of **O**bject **O**riented **P**rogramming

Here is another example,

The super Class bicycle will hold all the **common** characteristics namely apply brake, pedalling

Mountain bike has additional characteristics thick tyre.

Tandem bike has 2 seats & 2 handlebars.
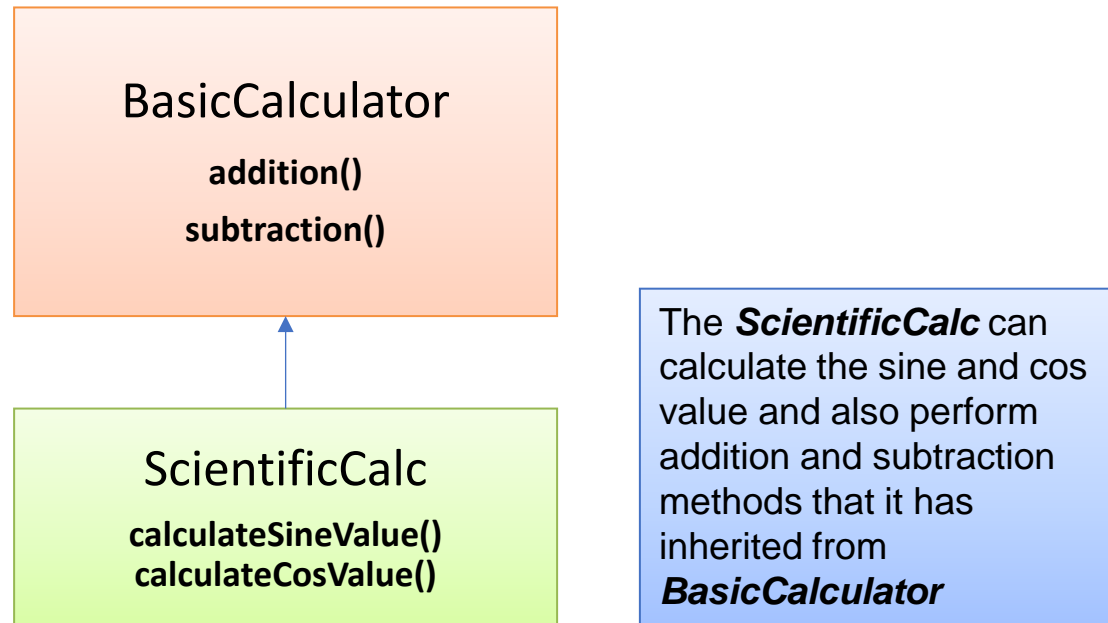
Road bike has shock absorbers.

Benefits of Inheritance

Benefits of Inheritance:

The primary benefit of inheritance is ***reusability***.

Once a behavior is defined in a super class, that behavior is automatically inherited by all its subclasses and reused. So developers need not redevelop the logic again.

Example:

```
┌─────────────────────────────┐
│      BasicCalculator        │
│                             │
│         addition()          │
│        subtraction()        │
└─────────────────────────────┘
              ▲
              │
┌─────────────────────────────┐
│       ScientificCalc        │
│                             │
│     calculateSineValue()    │
│     calculateCosValue()     │
└─────────────────────────────┘
```

The ***ScientificCalc*** can calculate the sine and cos value and also perform addition and subtraction methods that it has inherited from ***BasicCalculator***

Implementation

How to derive a subclass from parent class?

To derive a sub class, you use the ***extends*** keyword.

Assume you have a parent class called ***BasicCalculator***

Now you have to create another class called ***ScientificCalc*** which extends ***BasicCalculator*** so that you can inherit all methods of ***BasicCalculator***

```java
public class BasicCalculator {

    public int val1;
    public int val2;

    public int addition(){
        int sum = val1 + val2;
        return sum;
    }
    public int subtraction(){
        int diff = val1 - val2;
        return diff;
    }
}
```

```java
public class ScientificCalc extends BasicCalculator {

    public double calculateSineValue(){
        double sineValue = Math.sin(val1);
        return sineValue;
    }
    public double calculateCosValue(){
        double cosValue = Math.cos(val1);
        return cosValue;
    }
}
```

Features of Subclass

Features of the subclass:

▪ A subclass inherits all of the "**public**" and "**protected**" members of its parent, no matter what package the subclass is in.

▪ If the subclass is in the **same package**, then it also inherits the "**default**" access (members that do not have any access modifier) members of the parent

Assume A.java is the parent Class, B is a subclass present in same package and D is a subclass in a different package
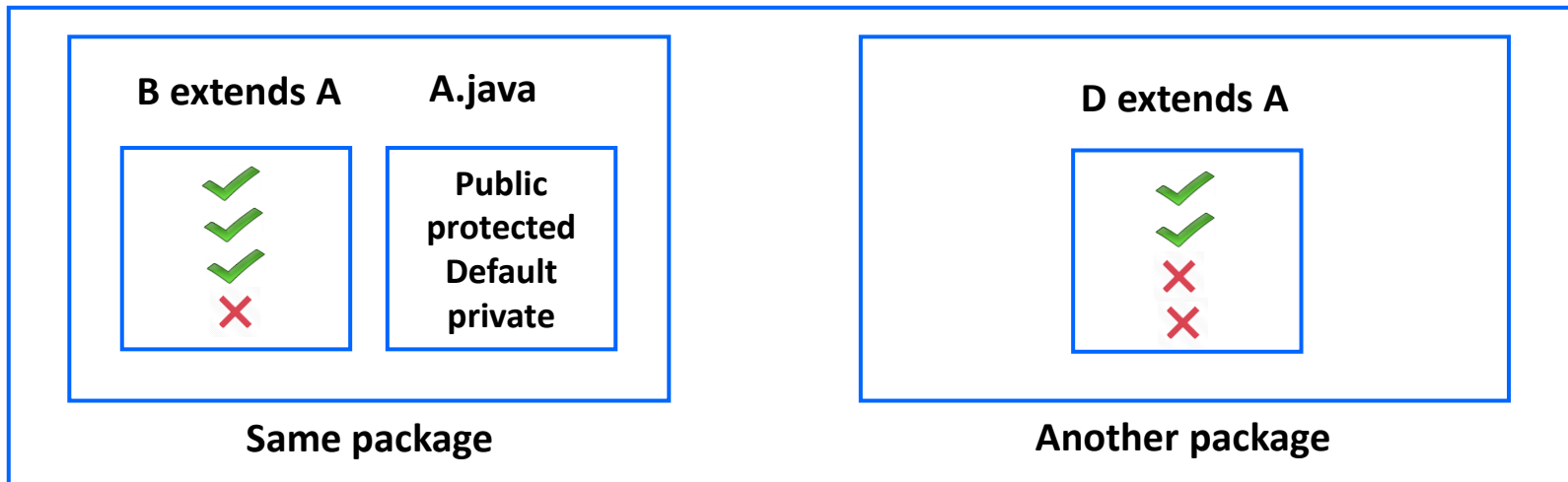
Inheriting fields in Subclass

- The inherited fields can be accessed **directly**, just like any other fields again based on the access modifiers.

- You can declare **new** fields in the subclass that are **not** in the **super** class.

- You can declare a **field** in the subclass with the **same name** as the one in the super class, thus hiding the parent fields (not recommended).

- A subclass does **not** inherit the **private members** of its parent class. However, if the super class has public or protected methods for accessing the private fields, these can also be accessed by the subclass.

Inheriting methods in a Subclass

• The inherited methods can be accessed **directly** just like any other methods in a class again based on the access modifiers.

• You can write a **new** instance of the **method** in the subclass that has the same signature as the one in the super class, thus **overriding** it.

• You can write a **new static** method in the subclass that has the same signature as the one in the super class, thus hiding it. Here, the super class method should also be static.

• You can declare **new methods** in the subclass that are **not** in the super class.

java.lang.Object

Object class is mother of all classes, residing in "*java.lang*" package.

- In Java language, all classes are subclass (extended) of the Object super class.

- Object class does not have a parent class.

- Object class implements behavior common to all classes including the ones that you write.

Following are some of the important methods of *Object* class,

- **getClass()** – Returns the runtime class of an Object.

- **equals()** – Compares two objects to check if they are the equal.

- **toString()** – Returns a String representation of the object.

Super class & Sub class

What is a super class?

Any class **preceding** a specific class in the class hierarchy is the super class. It is also called as **parent** class.

What is a sub class?

Any class **following** a specific class in the class hierarchy.

```
        Object
        /    \
   Class A   Class D
   /    \
Class B  Class C
```
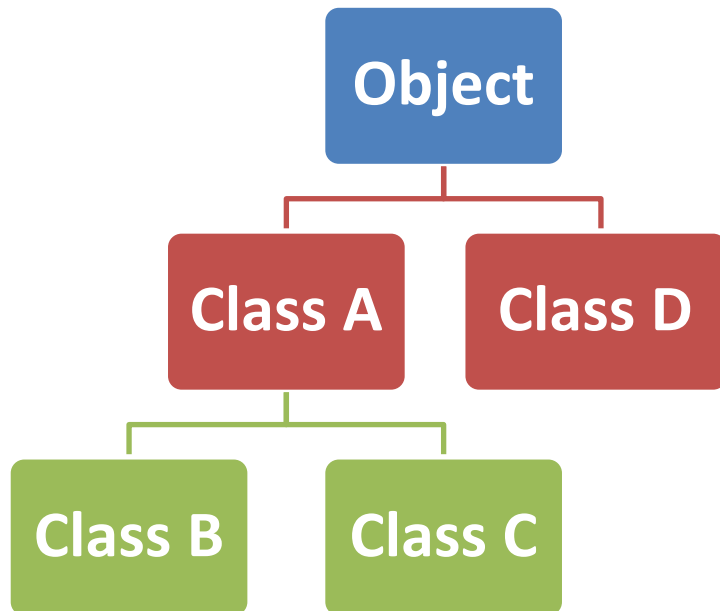
Class A is the **super class** for Class B and Class C

Class B and Class C are the **subclasses** of Class A

Object Class is the super class for all classes

Super - keyword

When is the super keyword used?

• The super keyword is used to access the variables and methods of super class

• If your method overrides one of it's super class methods, you can invoke the overridden method through the keyword super

Example:

```java
public class SuperClass {

    public void printMethod() {
        System.out.println("Superclass.");
    }
}
```

```java
public class SubClass extends SuperClass {

    // overrides printMethod in Superclass
    public void printMethod() {
        super.printMethod();
        System.out.println("Printed in Subclass");
    }
    public static void main(String[] args) {
        SubClass s = new SubClass();
        s.printMethod();
    }
}
```

Invokes the super method in the super class

super - method

When is the super constructor used?

• The super constructor call is used to call a constructor of its immediate super class

• Based on the argument passed, the corresponding constructor in the parent class is invoked

Few things to remember:

• The super() call must occur as the **first statement** in the constructor

• The super() call can **only** be used in **constructor** calls and **not method** calls.

Example – super constructor

Example: Create a main method and create an instance of TandemBike class to view output.

```java
public class Bicycle {

    public int wheels = 2;
    public int currSpeed = 150;
    int gear = 0;

    public Bicycle() {
        System.out.println("Inside Bicycle Constructor");
    }
}

public class TandemBike extends Bicycle {

    public int seats = 2;
    public int handlebars = 2;

    public TandemBike(){
        super();
        super.gear = 2;
        System.out.println("inside Tandem Bike Constructor"+gear);
    }
}
```

Super Class with default constructor

Super keyword to call the constructor in super class

Example - Inheritance

Problem Statement:

1. Create a class *BankAccount* with the following methods

- *depositMoney* – Prints the depositAmount.

- *withdrawMoney* – Prints the withdrawalAmount and calculates balance as mentioned below,

   balance = depositAmount – withdrawAmount

•The following instance variables need to be present in BankAccount

*withdrawAmount*, *depositAmount*, *interestRate* (defaulted to 9.5) and *balance*.

2. Create two subclasses *NRIAccount* and *SeniorCitizen* account which extends the *BankAccount* class

•Both the subclasses should have a method called *applyFixedDeposit* which should set the *interestRate* variable in the super class to 6.5 for *NRIAccount* and 10.5 for *SeniorCitizen* account and display the interest rate message.

3. Create a class *InheritanceDemo* with main method which creates instance of the NRIAccount and SeniorCitizen Account and invokes the methods *depositMoney(), withdrawMoney(), applyFixedDeposit()*

Overriding

Father is the super class
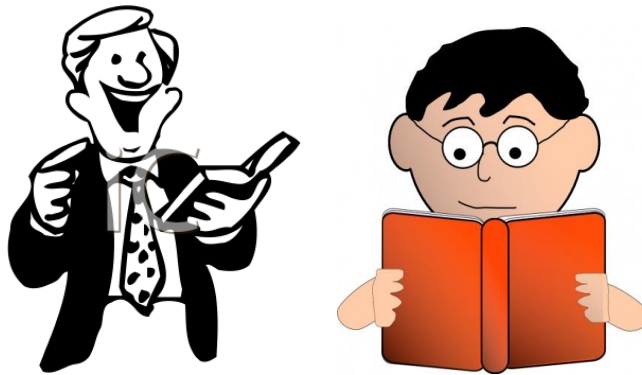
Son is the subclass and he inherits the properties of the super class (father)

Father likes to read books. His favorite genre is medical fiction

Son also likes to read books. His favorite genre is modern fantasy comics

Here, the son has inherited the father's behavior of reading books, but with a difference in the type of book. Here the son is overriding the behavior of father.

Method Overriding

What is method overriding?

Creating a method with the same name, arguments and return type in both the parent and the child class is called *method overriding*.

When do we override methods?

If a derived class (sub class) needs have a **different** method **implementation** from that of a super class, then that method can be overridden.

Example - Method Overriding

Parent Class:
```
public class Person{
    public String getName(){
        System.out.println("Parent: getName");
        return name;
    }
}
```

getName method in the super class, Person

Now, when you invoke **getName()** method of an object of subclass (**Student**) , output would be,
Student:getName

Child Class:
```
public class Student extends Person{
    public String getName(){
        System.out.println("Student:getName");
        return name;
    }
}
```

getName method is overridden in subclass Student

Run Time Polymorphism

What is polymorphism?

Polymorphism is the capability of a method to do different things based on the object used for invoking the method.

What is run time polymorphism?

Method overriding is an example of runtime polymorphism. Here, the JVM determines the method invocation at the runtime and not at the compile time. The method being invoked is based on the object which on which the method is triggered.

From the previous Example,

Person p = new Person();

Person s = new Student();

p.getName()// will print "Parent getName"

s.getName()// will print "Student getName"

Based on the object on which the method is invoked the appropriate **getName()** method is called.

Example – Run Time Polymorphism

Let us all understand run time polymorphism with an example !!

• Let us create a parent class, *Animal* and three subclasses, *Dog*, *Cow* and *Snake* which extends Animal

• All the classes should have a method, *whoAmI* which prints the appropriate class name as "I am a Dog" (or) "I am a Cow" etc.

• A new class, *RunTimePolymorphism* is created with a main method which creates an instance of all the classes created above and calls the *whoAmI* method for each instance.

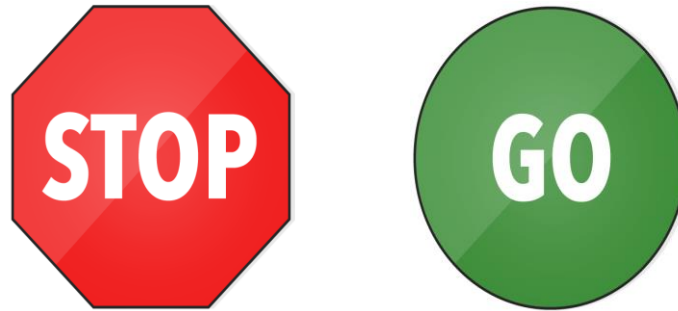# RuntimePolimorphism.java

```java
class Animal{
    void whoAmI() {
        System.out.println("I am a generic animal.");
    }
}
class Cow extends Animal{
    @Override
    void whoAmI() {
        System.out.println("I am a Cow.");
    }
}
class Dog extends Animal{
    @Override
    void whoAmI() {
        System.out.println("I am a Dog.");
    }
}
class Snake extends Animal{
    @Override
    void whoAmI() {
        System.out.println("I am a snake.");
    }
}
public class RuntimePolymorphism {
    public static void main(String[] args) {
        Animal obj = new Animal();
        obj.whoAmI();
        obj = new Cow();
        obj.whoAmI();
        obj = new  Dog();
        obj.whoAmI();
        obj = new Snake();
        obj.whoAmI();
    }
}
```

Console Output

```
I am a generic animal.
I am a Cow.
I am a Dog.
I am a snake.
```

There are 1 variable of type Animal. Only obj refers to instance of Animal class, all other refers to instances of subclass of Animal. From the output's result, we can see that the version of the method invoked is based on the actual object type

Time To Reflect

Trainees to reflect the following topics before proceeding.

- What is Inheritance?

- How to create a subclass?

- What is the super class for all classes?

- How to access the fields of the super class?

- What is overriding?

- What is run time polymorphism?

Thank you

You have successfully completed

Inheritance