While folks are joining

Get you laptops ready and login to your **replit** accounts.

We will be coding away in the session!

Crio Sprint : JAVA-2

Session 6

Today’s Session Agenda

- Polymorphism
- Method Overloading (Static Polymorphism)
- Method Overriding (Dynamic Polymorphism)

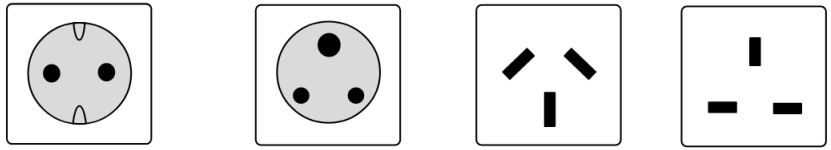
Why Polymorphism? - Scenario #1 Electric Socket

You don't want to pack all these in your travel bag either!

Why Polymorphism ? - Scenario #1 Electric Socket

Wouldn’t it will be better if we had sockets that could accept many different types of plugs.

Without Polymorphism



With Polymorphism

What is Polymorphism ?

- ***Polymorphism*** means having **many forms**.
- Perform a single action in different ways.
    - Define one interface & have multiple implementations.

Types of Polymorphism

- Compile Time Polymorphism
  - Method Overloading
  - Static Binding
- Runtime Polymorphism
  - Method Overriding
  - Dynamic Binding

Activity 1 - Addition

- Perform the addition of the given numbers. But user can enter any number of arguments.
- Possible Solution:
  - **addTwo(int, int)** method for two parameters
  - **addThree(int,int,int)** for three parameters
  - So on.
- What's the problem with the above technique?
  - Difficult to understand the behaviour of the method due to strange naming convention.
  - Difficult to track how many such methods are performing addition in the class due to different names.
- Can we avoid this problem?
  - Yes. Method Overloading.

Method Overloading

- What is Method Overloading?
  - Multiple methods having the **same name but difference in parameters**.
  - A class can **hold several methods** having the **same name.**
- **Three ways to overload methods:**
  - By changing the number of arguments/parameters.
  - By changing the data type of arguments.
  - By changing the Order of arguments.
- **Solution for Addition Activity**
    - **addition(int, int)**
    - **addition(int,int,int)**

1. By Changing the number of arguments / parameters

```
class SimpleCalculator
{
   int add(int a, int b)
   {
      return a+b;
   }
   int  add(int a, int b, int c)
   {
      return a+b+c;
   }
}
public class Demo
{
  public static void main(String args[])
  {
     SimpleCalculator obj = new SimpleCalculator();
     System.out.println(obj.add(10, 20));
     System.out.println(obj.add(10, 20, 30));
  }
}
```

Activity 2 - Find variations of Math.min()

- In Java's Math class, you will find many examples of overloaded methods.
- min() is overloaded with different data types.

`static double`	**min**`(double a, double b)` Returns the smaller of two `double` values.
`static float`	**min**`(float a, float b)` Returns the smaller of two `float` values.
`static int`	**min**`(int a, int b)` Returns the smaller of two `int` values.
`static long`	**min**`(long a, long b)` Returns the smaller of two `long` values.

3. By changing the Order of Arguments

```
class Student
{
   public void show(String name, int age)
   {
      System.out.println("Name of person = "+name+ " and age is = "+ age);
   }
   public void show(int age, String name)
   {
      System.out.println("Name of person = "+name+ " and age is = "+ age);
   }
   public static void main (String [] args)
   {
    Student s = new Student();
    // If student providing parameter of String and int  type then first method called
    s.show("Ram", 25);
    // If student providing parameter of int and String type then second method called
    s.show(25, "Ram");
   }
}
```

Curious Cats

- How does compiler recognize which overloaded method is called?
  - Compiler observers the signature of methods for multiple methods with same name.
  - Compiler understands that signatures are different and decides which appropriate method to call.
- Why is method overloading by changing the return type of a method, not possible?
  - Compiler only checks method signature for duplication and not the return type.
- When do we use Static Polymorphism?

Curious Cats

- Can we overload main() method in Java?
  - Yes, but JVM calls that main() method that receives string array as an argument only.
- Try running the below code:

```
public class MainMethodOverloadingTest
{
        public static void main(String[] args)
        {
          System.out.println("main(String[] args)");
          main();
        }
        public static void main()
        {
          System.out.println("main without args");
        }
        public static void main(String args)
        {
          System.out.println("main with string args");
        }
}
```

Summary - Method Overloading

- When a class has two or more than two methods which are having the **same name but different types of order or number of parameters,** it is known as Method Overloading.
- Method overloading is resolved during **compile time.**
- Three ways to overload methods:
    - By changing the **number of arguments/parameters.**
    - By changing the **data type of arguments.**
    - By changing the **Order of arguments.**
- Changing only return type with same parameters of method is not Method Overloading.

Activity 3 - Bank Interest Rates

Consider a scenario where Bank is a class that provides functionality to get the rate of interest. However, the rate of interest varies according to banks. For example, SBI and ICICI banks could provide 8% and 7% rate of interest.

What would be the output from this program?

- Interest rate will be printed as 5 for every bank.

What can we do to fix it?

- Create new methods in each bank which will give the expected rate.

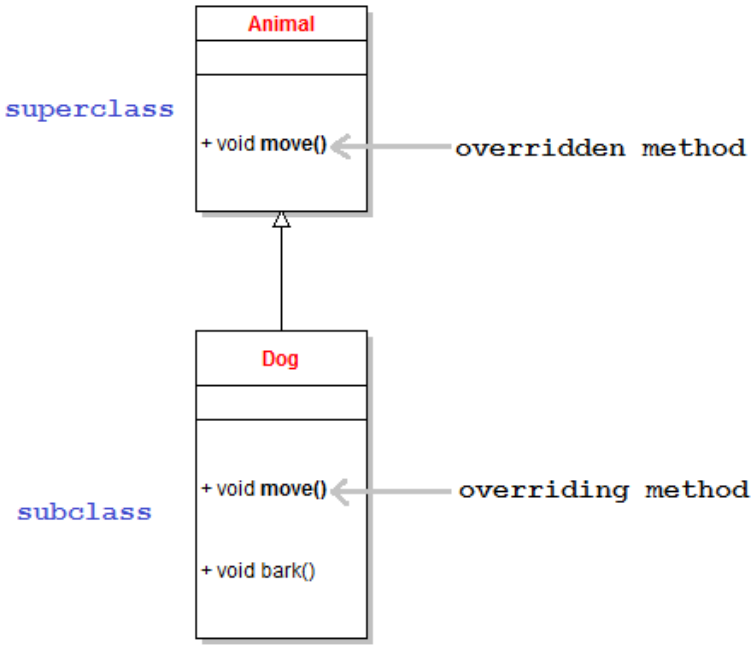
Can we use the same method name - *getRateOfInterest* in each bank subclass?

- Yes. Method Overriding.

```
class Bank{
  int getRateOfInterest(){return 5;}
}
//Creating child classes.
class SBI extends Bank{
}
class ICICI extends Bank{
}

class Test{
  public static void main(String args[]){
    SBI s=new SBI();
    ICICI i=new ICICI();
    System.out.println("SBI Rate of
Interest: "+s.getRateOfInterest());
    System.out.println("ICICI Rate of
Interest: "+i.getRateOfInterest());
  }
}
```

Method Overriding




```
class Bank{
  //Overridden Method
  int getRateOfInterest(){return 5;}
}
//Creating child classes
class SBI extends Bank{
  //Overriding Method
  @Override
   int getRateOfInterest(){return 8;}
}
class ICICI extends Bank{
   //Overriding Method
  @Override
   int getRateOfInterest(){return 7;}
}

class Test{
  public static void main(String args[]){
    SBI s=new SBI();
    ICICI i=new ICICI();
    System.out.println("SBI Rate of Interest"+ s.getRateOfInterest());
    System.out.println("ICICI Rate of Interest: "+i.getRateOfInterest());
  }
}
```

Summary - Rules for Method Overriding

1. **Only inherited methods** can be overridden.
2. The overriding method must have **same argument list**.
3. The overriding method must have **same return type**.
4. The overriding method **must not have more restrictive access modifier**.
    a. If the overridden method has *default* access, then the overriding one must be *default*, *protected* or *public*.
    b. If the overridden method is *protected*, then the overriding one must be *protected* or *public*.
    c. If the overridden method is *public*, then the overriding one must be only *public*.

How to call an Overridden Method?

Suppose Base b = new Derived();

what is the result of the call b.methodOne();?

- A subclass might need to call the parent method for some operation to be successful.
- But the parent method is overridden, so how can we still call it?
- Use ***super.method()*** to force the parent's method to be called.
  - Replit

```
public class Base {
  public void methodOne(){
   System.out.print("A");
    methodTwo();
  }
  public void methodTwo(){
   System.out.print("B");
  }
}
public class Derived extends Base {
 @Override
 public void methodOne(){
   super.methodOne();
   System.out.print("C");
  }
 @Override
  public void methodTwo(){
   5. System.out.print("D");
    super.methodTwo();
     }
}
class Main {
 public static void main(String[] args) {
  Base b = new Derived();
         b.methodOne();
 }
}
```

Curious Cats

- Can we override a static method?
  - No, static methods cannot be overridden in Java.
  - Static methods are class-based and are called by class directly.
  - They don’t need objects to be invoked at runtime.
  - Hence the static method dispatch is determined by the compiler.
- Can we override final method?
  - A final method means that it cannot be re-implemented by a subclass, thus it cannot be overridden.
- Can we override constructor?
  - No, we cannot override a constructor.
  - Subclasses cannot override a parent class’s constructor as a constructor of two classes cannot be the same.

Curious Cats

- Do we really need to use @Override annotation?
  - Not really but good to have.
  - Makes it human readable to understand that the method is a overriding method.
  - It helps to catch bug at compile time with less effort.

Activity 4 (Optional)

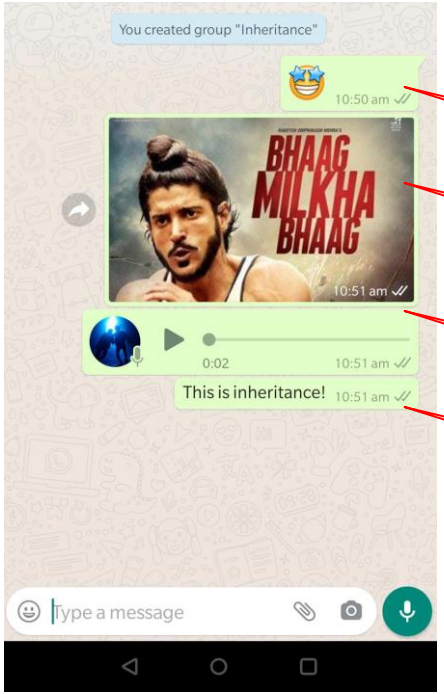
- NumberGame - Replit
- Solution: NumberGameSolution

Polymorphism Byte Overview

Messaging Application

Recap - YouChat - Messaging Platform

Now Support different kind of messages



Emoji

Image

Voice

Text

New Requirements

Text Message

Check Validity - if length of text is < 100

Image Message

Check Validity - if image is not empty

Possible Solution

- You currently have the required methods in all the different message types.
- Add validation methods in each type of message class and perform validation logic

```
public class TextMessage extends Message {
  //other methods
  public boolean isValidTextMessage(){
    if(this.getTextMessageContentSize() > 100){
      return true;
    }
    return false;
  }
}
```

```
public class ImageMessage extends Message {
  //other methods
public  boolean isValidImageMessage(){
    if(this.getImageMessageContent() != null){
      return true;
    }
    return false;
  }
}
```

  - What's the issue with the above approach?
    - Clients (e.g. AndroidHandler.java) need to be aware of the method names used by each of the message types.
    - Every time a new message type is introduced or teams want to have their own implementation, the clients would need to make code changes.

Polymorphism Based Solution

```
public abstract class Message {
   // other methods and fields
   public abstract boolean isValid();
}
public class TextMessage extends Message {
   //other methods
   @override
   public boolean isValid(){
      if(this.getTextMessageContentSize() > 100){
         return true;
      }
      return false;
   }
}
public class ImageMessage extends Message {
   //other methods
  @override
  public  boolean isValid(){
      if(this.getImageMessageContent() != null){
         return true;
      }
      return false;
   }
}
```

Why is this a better solution?

- Each message type class overrides the default base class functionality.
- Clients are not impacted

Take home exercises for the session

- You will explore **Polymorphism** with this real world scenario in the following Byte:
  - Polymorphism Byte - Crio.do
- Complete the Quiz Activity (details on the google site) for Session 6.
  - Java 2 - Session 6 Quiz

These details are also available on the site.

Feedback

Thank you for joining in today.

We’d love to hear your thoughts and feedback - Feedback for JAVA-2 Session

Further Reading

- Java - When NOT to call super() method when overriding? - Stack Overflow

Thank you

Why Polymorphism?

- Reusability
- Flexibility
- Extensibility