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 1

Contents of JAVA-2 Sprint

- Introduction to Java
- OOPS in Java
  - Static and final
  - Access Modifiers
  - Encapsulation and Abstraction
  - Inheritance, Composition and Polymorphism
  - Interfaces and Abstract classes
  - Overloading and Overriding
- Micro Experience - QCalc
- Buildout - XURL

Today’s Session Agenda

- Recap - Class in Java
- Constructors
  - Default Constructor
  - Parameterized Constructor
  - Constructor Overloading
  - Constructor Chaining
  - Copy Constructor

Recap - Class in Java

- A **class** is a template from which individual objects are created.
- A class contains **fields** and **methods**.
- A **object** is an instance of a class. It has 3 characteristics:
  - State - Represents data of an object.
  - Behaviour - Represents the functionality of an object.
  - Identity - Assigned by JVM to identify each object uniquely.
- How to create an instance of a class ? - **new()** operator.
- For example - There are many bicycles (objects) available in the market with a same model (class) but with different colors.

```
class Bicycle {

    int speed = 0;
    int gear = 1;
    String color;

    void setColor(String color){
        this.color = color;
    }

    void changeGear(int newValue) {
        gear = newValue;
    }

    void speedUp(int increment) {
        speed = speed + increment;
    }

    void applyBrakes(int decrement) {
        speed = speed - decrement;
    }

}
```

How is the object created?

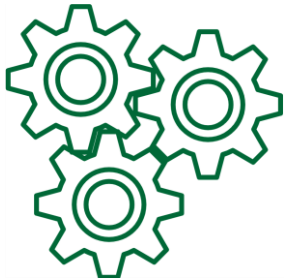
- **Declaration of a reference variable**
  - **Bicycle mycycle = new Bicycle();**
- **Create an object**
  - **Bicycle mycycle = new Bicycle();**
- **Assign the object to the reference variable**
  - **Bicycle mycycle = new Bicycle();**

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

      // Create two different
      // Bicycle objects
      Bicycle bike1 = new Bicycle();
      Bicycle bike2 = new Bicycle();

   }
}
```

- **Are we invoking a method named Bicycle() here?**
  - **No, we're invoking a Bicycle Constructor.**

What is a constructor? Why do we need it?

- A **constructor** is a special method that **initializes new objects** or *instances* of the class.
- Without a constructor, you cannot create instances of the class.
- **Called when an instance of the class is created.** Memory for the object is allocated.
- Constructors always have the **same name as the class**.
- It must have **no explicit return type.**
- Types of constructors:
  - Default Constructor (no-arg constructor)
  - Parameterized Constructor



Default Constructor

- A constructor is called **Default Constructor** when it doesn't have any parameter.
- Syntax: `<class_name>(){ }`

- What is the purpose of a default constructor?
  - It is used to provide default values to the object fields like 0, null, etc., depending on the type.

```
class Company {
   String name;
   // default constructor
    public Company() {
      name = "Crio.Do";
   }
}

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

    // object is created in another class
    Company obj = new Company();
    System.out.println("Company name = "
+ obj.name);
   }
}
```

Curious Cats

- What happens if we don't declare a constructor in the class?
  - If there is no constructor in a class, Java compiler automatically creates a default constructor.
- Can we have a non constructor method with the same name as the class?
  - Yes, if the methods have a return type. Constructors cannot have a return type.
  - But, this is not a good practice.

Curious Cats

What makes a constructor different from regular class methods?

Method	Constructor
It can have any user defined name.	It must have it’s class name.
It should have a return type.	It doesn’t have any return type.
It is called explicitly either with object reference or class reference.	It is called automatically whenever an object is created.
Compiler does not provide any method by default.	Java Compiler provides a default constructor if not defined.
It describes behaviour of an object.	It is used to initialize the state of an object.

Activity 1 - What is missing here? How to improve it?

```
class Rectangle {
  double length;
  double breadth;
  public Rectangle() {
    length = 0;
    breadth = 0;
  }
  public void setLength(double length){
    this.length = length;
  }
  public void setBreadth(double breadth){
    this.breadth = breadth;
  }
  public double calculateArea(){
    return length * breadth;
  }
}
```

```
class Main {

  public static void main(String[] args) {

    System.out.println("hello world");
    Rectangle r1 = new Rectangle();
    r1.setLength(10.0);
    r1.setBreadth(20.0);
    System.out.println(r1.calculateArea());
    Rectangle r2 = new Rectangle();
    System.out.println(r2.calculateArea());

  }
}
```

Parameterized Constructor

- It has a specific number of parameters.
- It is used to provide different values at initialization.
- Check for correctness upon object construction and disallow any objects from being in a invalid state.
- Tryout this example on replit
    - Rectangle

```
class Rectangle {
  double length;
  double breadth;

  public Rectangle(double length, double breadth) {
    if(length >= 0 && breadth >= 0){
      this.length = length;
      this.breadth = breadth;
    } else{
      System.out.println("length & breadth should be > 0");
    }
  }

  public double calculateArea(){
    return length * breadth;
  }
}

class Main {
  public static void main(String[] args) {
    System.out.println("hello world");
    Rectangle r1 = new Rectangle(10.0, 20.0);
    System.out.println(r1.calculateArea());
    Rectangle r2 = new Rectangle(10.0,-10.0);
    System.out.println(r2.calculateArea());
  }
}
```

Curious Cats

- Does compiler always make a no-arg constructor even if the parameterized one is present?
  - No. The compiler gets involved in constructor making only if there is no constructor declared in class.
  - If you have a parameterized constructor declared, and still want a no-arg constructor, you'll have to add another declaration for it.

Activity 2.1 - Implement Counter Class

- Implement a class called **Counter**.
- The class **contains a number** whose value can be incremented and decremented.
- **Set** the start value of the number to **0**.
- And the following methods:
  - `public int value()` returns the current value of the counter
  - `public void increase()` increases the value by 1
  - `public void decrease()` decreases the value by 1

Activity 2.1 - Implement Counter Class

```
class Counter{
  int count;
  public Counter(){
    count = 0;
  }
  public int value(){
    return count;
  }
  public void increase(){
    count++;
  }
  public void decrease(){
    count--;
  }
}
```

Activity 2.2 - Implement Counter Class

New Requirement

- The start value of the counter should be set with a specific **startValue**.
    - startValue may range from 0 to 1000 and beyond.

Activity 2.2 - Implement Counter Class

```
class Counter{
  int count;
  public Counter(int startValue){
    if( startValue >= 0 ){
      this.count = startValue;
    }
    else{
                System.out.println("Counter must be  >= 0");
    }
  }
  public int value(){
    return count;
  }
  public void increase(){
    count++;
  }
  public void decrease(){
    count--;
  }
}
```

Activity 2.3 - Implement Counter Class

Yet Another Requirement

- **Each Counter instance should be assigned a name.**

Activity 2.3 - Implement Counter Class

```
class Counter{
  int count;
  String name;

  public Counter(int startValue, String name){
    if( startValue >= 0 ){
      this.count = startValue;
    } else{
     System.out.println("Counter must be >= 0");
    }
    this.name = name;
  }

  public int value(){
    return count;
  }

  public void increase(){
    count++;
  }

  public void decrease(){
    count--;
  }
}
```

Did you notice anything strange?

- **We are modifying the constructor's parameters list with each new requirement.**

Can we improve upon this and avoid changing the constructor signature every time?

Constructor Overloading

- **Constructor Overloading** means having more than one constructor in a class.
- Overloaded constructors must have different argument lists (but no return type).
- For e.g. Color Class in java.awt package

Constructor Summary

Constructors

Constructor and Description
`Color(ColorSpace cspace, float[] components, float alpha)` Creates a color in the specified `ColorSpace` with the color components specified in the `float` array and the specified alpha.
`Color(float r, float g, float b)` Creates an opaque sRGB color with the specified red, green, and blue values in the range (0.0 - 1.0).
`Color(float r, float g, float b, float a)` Creates an sRGB color with the specified red, green, blue, and alpha values in the range (0.0 - 1.0).
`Color(int rgb)` Creates an opaque sRGB color with the specified combined RGB value consisting of the red component in bits 16-23, the green component in bits 8-15, and the blue component in bits 0-7.
`Color(int rgba, boolean hasalpha)` Creates an sRGB color with the specified combined RGBA value consisting of the alpha component in bits 24-31, the red component in bits 16-23, the green component in bits 8-15, and the blue component in bits 0-7.
`Color(int r, int g, int b)` Creates an opaque sRGB color with the specified red, green, and blue values in the range (0 - 255).
`Color(int r, int g, int b, int a)` Creates an sRGB color with the specified red, green, blue, and alpha values in the range (0 - 255).

Activity 3 - Implement Improved Counter Class

Implement all the requirements mentioned below using constructor overloading.

- **By default, set the start value of the counter to 0 and name as "Counter One".**
- **Support a way to initialize the start value of the counter to a specific startValue.**
  - **startValue may range from 0 to 1000 or beyond.**
  - **name should be "Counter One".**
- **Support a way to initialize the counter to 0 and have a name.**
- **Support a way to initialize the counter with startValue and name.**

Activity 3 - Implement Improved Counter Class

```
class Counter{
 int count;
 String name;

 public Counter(){
   count = 0;
   name = “Counter  One”;
 }

 public Counter(int startValue){
    if( startValue >= 0 ){
     this.count = startValue;
   } else{
     System.out.println("Counter must be >= 0");
   }
 name = “Counter  One”;
 }

 public Counter(String name){
  this.name = name;
  count = 0;
 }

 public Counter(int startValue, String name){
    if( startValue >= 0 ){
     this.count = startValue;
   } else{
     System.out.println("Counter must be >= 0");
   }
  this.name = name;
 }

 public int value(){
  return count;
 }

 public void increase(){
  count++;
 }

 public void decrease(){
  count--;
 }
}
```

Constructor Chaining

- Calling a constructor from the another constructor of same class using **this**.
- **this** should be the first line of the constructor block.
- Compare the previous implementation of Counter class with the current one. What do you observe?
- What is the purpose of constructor chaining?

```
class Counter{
  int count;
  String name;

  public Counter(){
    this("Counter  One");
  }
  public Counter(int startValue){
    this(startValue,"Counter One" );
  }
  public Counter(String name){
    this(0, name);
  }
  public Counter(int startValue, String name ){
    this.name = name;
    if( startValue >= 0 ){
      this.count = startValue;
    } else{
     System.out.println"Counter must be>= 0");
    }
  }
  public int value(){
    return count;
  }
  public void increase(){
    count++;
  }
  public void decrease(){
    count--;
  }
}
```

Activity 4.1 - Copying an Object in Java

- How would you make a copy of an object in Java?
  - Let's try it out for this example - Spotify AudioFeatures
- Possible Solution
  - Spotify AudioFeatures Solution
  - What's the issue with this approach?
    - Too many parameters to copy.
    - Frequent Repetition of code; if required at multiple places.
- Any alternative approach?
  - Yes. Use Copy Constructor

Copy Constructor in Java

- A **special type of constructor that creates an object using another object** of the same Java class.
- It is a **one-argument** constructor.
- Used to **create a copy of an object that has too many fields.**
- For a class *Student*, a copy constructor can be declared as follows:
    - Student(Student s) {
      // copy constructor code;
      }

Complex Number - Copy Constructor

```
class Complex {

  private double re, im;
  // A normal parameterized constructor
  public Complex(double re, double im) {
    this.re = re;
    this.im = im;
  }

  // copy constructor
  public Complex(Complex c) {
    System.out.println("Copy constructor called");
    re = c.re;
    im = c.im;
  }

  public String print() {
    return "(" + re + " + " + im + "i)";
  }
}
```

```
public class Main {

  public static void main(String[] args) {
    Complex c1 = new Complex(10, 15);

    // Following involves a copy constructor call
    Complex c2 = new Complex(c1);

    // Note that the following doesn't involve a copy constructor call
    // as non-primitive variables are just references.
    Complex c3 = c2;

    System.out.println(c2.print()); // toString() of c2 is called here
  }
}
```

Activity 4.2 - Copying an Object in Java

Make a copy of an AudioFeatures Object using Copy Constructor

Spotify AudioFeatures

Solution

- Spotify AudioFeatures Solution

Take home exercises for the session

- JAVA-2 Session 1 Quiz
- **Takehome I - Replit**
- **Takehome II - Replit**

Feedback

Thank you for joining in today.

We'd love to hear your thoughts and feedback -
https://forms.gle/N6QCEFYqzZqEg1jXA

Further Reading

- Private Constructor in Java: Use Cases Explained with Example | upGrad blog
- **Java Program to implement private constructors (programiz.com)**
- **Java Copy Constructor | Baeldung**

References

- Head First Java: A Brain-Friendly Guide, 2nd Edition (Covers Java 5.0) Book
- Learn Java Programming (programiz.com)
- Java Programming Language - GeeksforGeeks

Thank you