1
2
3
@psychiatricmind
4
5
6
7
8
9

Crio Sprint: JAVA-112

Session 3 - OOPs: Encapsulation

Today’s Session Agenda

- Classes, Objects and Constructors
- Object Oriented Programming (OOP)
- 4 Pillars of OOP
- Encapsulation
- Access modifiers
- Getters and Setters

Recap - Classes, Objects & Constructors

What are objects?

Well, Objects could be:

Something Visible

Something you Can't Touch

Account, Contest, Post, Comment

All objects have …

- **Identity**: John's Bicycle
- **State(Attributes)**: color, speed, gear
- **Behaviours**: switchGear(), applyBrakes(), speedUp(), stop()

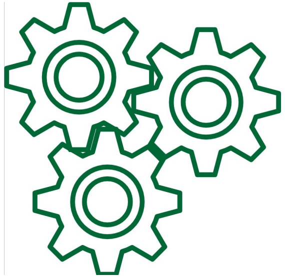
Recap - Class in Java

- 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 related to an object in memory.
    - 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.
- User of the **“.”** operator to access Object Fields or Methods

```
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;
    }

}
```

What is a constructor? Why do we need it?

- **Constructor** - special method that **initializes new objects/***instances* of class.
- Without a constructor, you cannot create instances of the class.
- It is **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
- A class can have **multiple constructors** (we will revisit this when we get to Polymorphism)



Constructor examples

```
class Company {
     String name;

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

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

     // object being created here
     Company obj = new Company();
     System.out.println("Company name = " +
obj.name);
     }
}
```

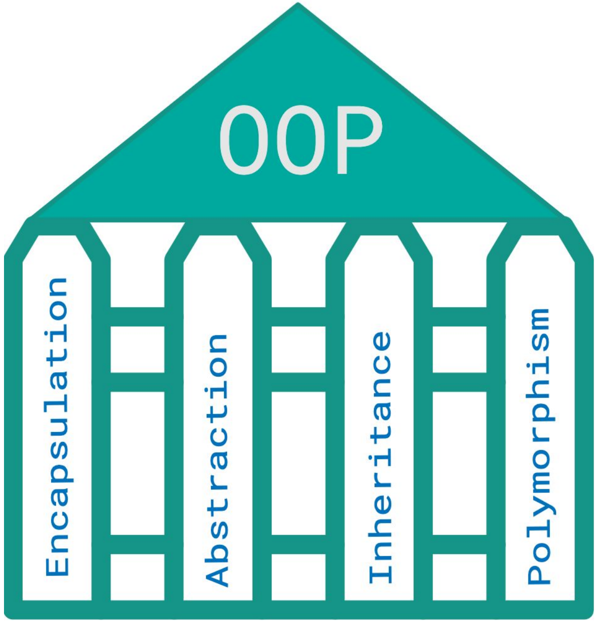
```
class Rectangle {
   double length;
   double breadth;

   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());
     }
}
```

Object Oriented Programming (OOP)

Why Object Oriented programming?

- Effective Problem Solving
- Modularity
- Reusability
- Flexibility
- Testability

Four Pillars of OOP



Goals

- **Encapsulation**
  - Reduce Complexity + Data Security
- **Abstraction**
  - Hide Complexity + Isolate Impact of changes
- **Inheritance**
  - Eliminate Redundant Code + Reusability
- **Polymorphism**
  - An object can take many forms

Encapsulation in Real World - Scenario #1 Restaurant

- Have you ever had dinner at a restaurant?
- What are the things you do when you are at a restaurant?
- Can you change the price of the dish items displayed on the menu card?
- Can you enter the kitchen and start making your favourite dish?
- Can you take orders from another table and ask waiter to stand aside?
- Can you add / remove cash from the Manager’s cash register?

Need for Encapsulation

Suppose you have an account in the bank.

The bank account Class is represented below:

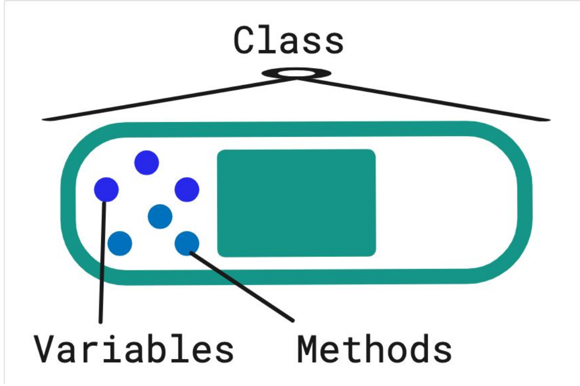
```
class Account {
 public double balance;
 public int accountNumber;
 public void deposit(double a){
   balance = balance + a;
 }
 public void withdraw(double a){
   balance = balance - a;
 }
}
```

Can you figure out what could go wrong if this solution is used?

What do we accomplish with these changes?

```
class Account {
 private double balance;
 private int accountNumber;

 public void deposit(double a){
   if ( a <= 0  ){
     System.out.println(“a should be > 0”);
     return;
   }
   balance = balance + a;
 }
 public void withdraw(double a){
   if ( a <= 0  ){
     System.out.println(“a should be > 0”);
     return;
   }
   if (balance - a < 0 ){
     System.out.println(“Insufficient funds”);
     return;
   }
   balance = balance - a;
 }
}
```

What is Encapsulation ?



- **Binding the data and related methods into a single unit**.
- Keeps the data and methods safe from external interference
  - **Data Hiding**
- Characteristics of Encapsulated Code:
  - Others knows how to access it and what can be accessed.
  - Can be easily used regardless of the internal implementation details.
  - There is no side effect of this code on the rest of the application.

Java Collections is the good example of encapsulated code

- We can insert and retrieve the data using provided methods.
- How and where the data is actually stored is hidden from the user.
- We can easily use Collections without bothering about it's implementation.

Curious Cats

- What’s the relationship between **Encapsulation** and **Data Hiding**?
  - Think about this - If all the data fields and methods in a class are public, that exhibits encapsulation, but not data hiding.
  - So, Encapsulation enables Data Hiding, but they are not the same!
  - Data Hiding is achieved by using Access Modifiers.

How do we achieve Data Hiding?

- By using Access Modifiers (Access Specifiers)
- By using Getters and Setters

Java Access Modifiers

Java Access Modifiers

- It is used to set the accessibility of **classes**, **constructors**, **methods**, and other **members** in Java.
- There are four access specifiers keyword in Java.
  - default (No keyword required)
  - **public**
  - **private**
  - **protected** (Will be discussed during inheritance), is slightly different from default
- What do we mean by a class or member(attribute) or method being **visible** or **accessible**?
  - For a class - it means an object of this class can be created in another class
  - For a member or method - it means the member or method (on an object of this class) can be referred to or invoked by another class.
  - We will look at code examples in the coming slides to understand this.

Public Access Modifier

- A **public** access modifier is a modifier that does not restrict the members at all.
- A **public** member (method or field) is accessible within the package as well as outside the package. Basically, everywhere!
- A package is a **namespace that organizes a set of related classes**.

```
class A {
    public int a;
    public void display() {
        System.out.println("Crio.Do!!");
    }
}
public class Main {
    public static void main(String args[]) {
        A obj = new A ();
        obj.display();
        obj.a = 5;
    }
}
```

Private Access Modifier

- The **private** access modifier is the one that has the lowest accessibility level.
- The scope of private entities (methods or fields) is **limited to the class in which they are declared**.
- Can you declare a **constructor** as **private**?
  - A constructor can be declared as private but you cannot create an object of the class from anywhere!

What will be the output?

```
class TestClass {
    //private variable and method
    private int num=100;
    private void printMessage() {
      System.out.println("Hello java");}
}

public class Main {
 public static void main(String args[]) {
   TestClass obj=new TestClass();
   System.out.println(obj.num);
   obj.printMessage();
   }
}
```

Curious Cats

- What’s the disadvantage of making all fields and methods **public**?
- What fields would you make **public**?
- What methods would you keep **private**?
- Did you notice that class can also be public or private?
    - When would you create a **private** class?

Default Access Modifier

- A **default** access modifier in Java has no specific keyword.
- **Whenever** the **access modifier is not specified**, then it is assumed to be the **default**.
- Default members are accessible only inside the **package**.

```java
class BaseClass
{
   void display()     //no access modifier indicates default modifier
     {
        System.out.println("BaseClass::display() with 'default' scope");
     }
}

class Main
{
   public static void main(String args[])
     {
       //access the class with default scope
       BaseClass obj = new BaseClass();

       obj.display();    //access class method with default scope
     }
}
```

How do getters and setters help in Data Hiding?

- **Getters** *(accessors)* and **setters** *(mutators)* allow you to **control how important variables are accessed and updated** in your code.
- Setters **Validate** input, before setting the variable values.
- Read member variable only through Getters.

- Are simple getters and setters enough to achieve Data Hiding?
  - No.
  - Let’s see how we can achieve it.

```
class Number {
    private int number;

    //Properly validated Setter
    public void setNumber(int number) {
      if (number < 1 || number > 10) {
      // Print error
    }
      this.number = num;
    }

    //Getter
    public int getNumber() {
      return this.number;
    }
}
```

How to achieve Proper Encapsulation?

- Would you allow anyone on the internet to deduct money from your bank account?
  - **Restrict access**
    - Keep data members private.
    - Keep methods private which need not be accessed from outside.
    - Create public methods to control access of object’s data from outside classes/applications.

- Can a week have more than 7 days?
  - **Know the bounds of values**
    - Be aware of valid values for each data member.

- Can rectangle have a length and breadth both zero?
  - **Initialize data elements to valid initial values** for an empty/new object using default/parameterized constructor.

How to achieve Proper Encapsulation?

- Does it make sense to represent your name using Integer?
    - **Choose the data types wisely**.
        - Choose data types that are appropriate to hold valid values.

- Can we change the time to Negative value?
    - **Validate input** before changing the data values stored in the object.

- Finally!
    - Double check all operations that change the data to maintain its validity.

Activity 1.1 - CustomTime Class

```
public class CustomTime {
   int hour;
   int minute;
   int second;
   void setTime(int newHour, int newMinute, int
newSecond)
      { /* mutator implementation */ }
   int[] getTime()
      { /* accessor implementation */ }
   void incrementTime()
      { /* mutator implementation */ }
};
```

Current Implementation

1. Compile and run the program.
2. Look at the output. Does it make sense? Why or why not?

Activity 1.2 - CustomTime Class

1. Add the following lines to Main.java just before the end of the main method:

```
currTime.hour = 31;
currTime.minute = -10;
currTime.second = 450;
temp = currTime.getTime();
hr = temp[0];
min = temp[1];
sec = temp[2];
System.out.println(
"After direct assignment, the current time is: "
+ hr + ":" + min + ":" + sec
);
```

2. Compile and run the program.
3. Look at the new output. Does it make sense? Why or why not?

4. We need to fix the problem caused by declaring the data in the CustomTime class as public.

5. Change CustomTime.java to make the 3 data declarations private. Compile the program. What happens? Why?

6. Remove the lines that were added to Main.java in step 1 above.

7. Compile and run the program.

Activity 1.3 - CustomTime Class

1. Change the call *currTime.setTime(20, 15, 43);* in Main.java to the following: *currTime.setTime(-55, 99, 1025);*
2. Compile and run the program.
3. Look at the new output. Does it make sense? Why or why not?

Activity 1.4 - CustomTime Class

1. Let's fix the setTime() method.

```
void setTime(int newHour, int newMinute, int newSecond) {
      if (newHour >= 0 &&  newHour <= MAX_HOURS) {
         hour = newHour;
      }
      else {
         System.out.println("Error: hour must be between 0 and 23 inclusive");
         hour = 0;
      }
      if (newMinute >= 0  && newMinute <= MAX_MIN_SECS) {
         minute = newMinute;
      }
      else {
         System.out.println("Error: minute must be between 0 and 59 inclusive");
         minute = 0;
      }
      if (newSecond >= 0 && newSecond <= MAX_MIN_SECS) {
         second = newSecond;
      }
      else {
         System.out.println("Error: second must be between 0 and 59 inclusive");
         second = 0;
      }
}
```

2. Compile and run the program.

3. Why is this version of the setTime() method more secure than the previous version?

4. Look at the new output. Does it make sense? Why or why not?

5. Change the call to *currTime.setTime(20, 15, 43);* in Main.java to the following:
currTime.setTime(23, 59, 59);
Compile and run the program.

6. Look at the new output. Does it make sense? Why or why not?

5 minute break

Activity 1.5 - CustomTime Class

1. Add an appropriate constructor to the Time class.
2. What values should be used to initialize hour, minute, and second in the constructor? Why are these times appropriate?
3. Compile and run the program.

Activity 1.6 - CustomTime Class

1. Change the call to *currTime.setTime(23, 59, 59);* in Main.java to the following: *currTime.setTime(20, 15, 43);*
2. Let's fix the incrementTime() method

```
void incrementTime () {
    second = ++second % (MAX_MIN_SECS + 1);
    if(second == 0) {
      minute = ++minute % (MAX_MIN_SECS + 1);
    }
    if(second == 0 && minute == 0 ) {
      hour = ++hour % (MAX_HOURS + 1);
    }
}
```

3. Compile and run the program.

4. Look at the new output. Does it make sense? Why or why not?

5. Why is this version of the incrementTime() method more secure than the original version?

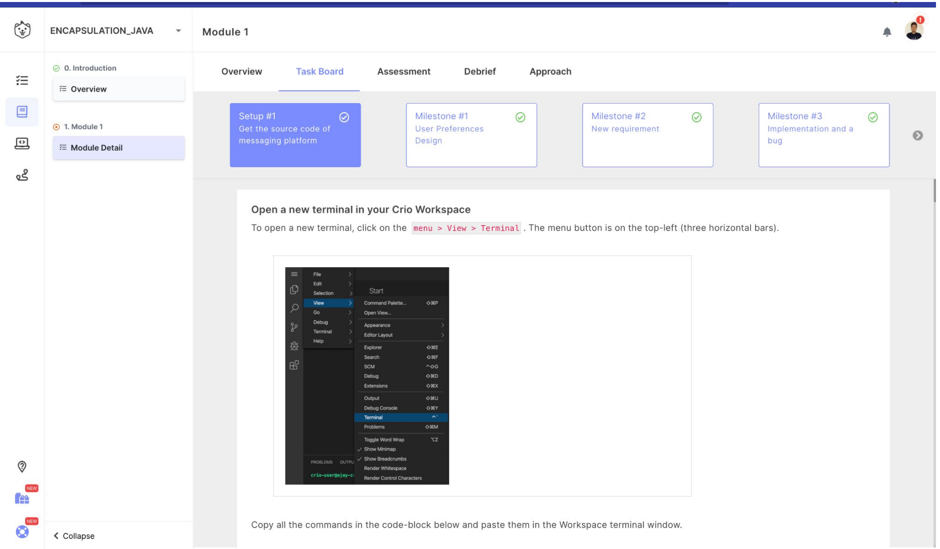
Encapsulation Exercises Byte Overview

What is a Byte?

- Bytes help you learn a specific concept or tool from the basics(eg: Encapsulation, Inheritance) in a self-paced manner
- Bytes contain activities to give you practice all the while learning new skills and *can involve some level of self-exploration* using given references to solve these activities
- Explanations are given for these activities for you to compare your findings and get any additional related context
- These usually takes around 2-4 hours to complete

Platform Introduction - Encapsulation Byte - Crio.do

Crio Byte

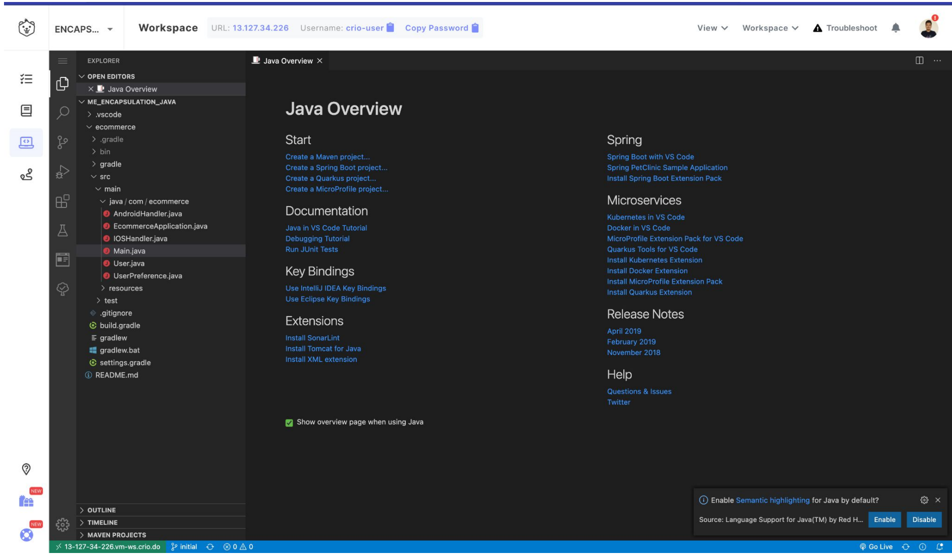


Open a new terminal in your Crio Workspace

To open a new terminal, click on the `menu > View > Terminal` . The menu button is on the top-left (three horizontal bars).

Copy all the commands in the code-block below and paste them in the Workspace terminal window.

Workspace



Enable Semantic highlighting for Java by default?

Source: Language Support for Java(TM) by Red H...

Let’s Solve Elementary Exercise - Custom Time

Let's Solve Elementary Exercise - Custom Time

Overview: Reinforcement Exercise - WhatsApp Profile

Reinforcement Exercise - WhatsApp Profile

Overview: Challenge Exercise - Snake

Challenge Exercise - Snake

Take home exercises for the session

- Encapsulation Byte
  - Encapsulation Quiz (Link Present in Byte)

All of these details are also available on the site.

Questions

1. What are the four main principles of object-oriented programming (OOP)?
2. Explain the concept of encapsulation in object-oriented programming and provide an example along with its benefits.
3. Explain the purpose and importance of using getters and setters in object-oriented programming. Provide an example to demonstrate their usage and benefits.
4. What are access modifiers in Java? Provide examples for each type.

Session Revision Quiz

Quiz Link

Solve this quiz to access your understanding of session’s topics clearly

Further Reading

- Java Quiz 15: Improve Encapsulation of Your Code - DZone Java
- Bounding Box (Optional Assignment)

References

- OOPs in Java: Encapsulation, Inheritance, Polymorphism, Abstraction (beginnersbook.com)
- What is “Encapsulation” and What are the Benefits of It?

Thank you