# 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 2

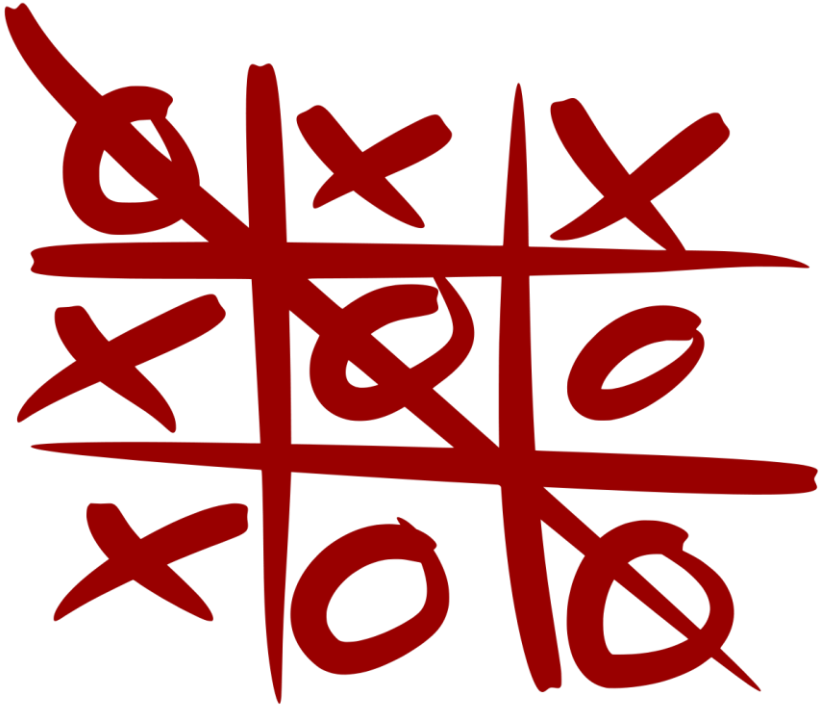
# Today’s Session Agenda

- Four Pillars of OOP
- Getters and Setter
- Activity - Implement CustomTime Class

# Why Object Oriented programming?

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

# Program for Tic Tac Toe

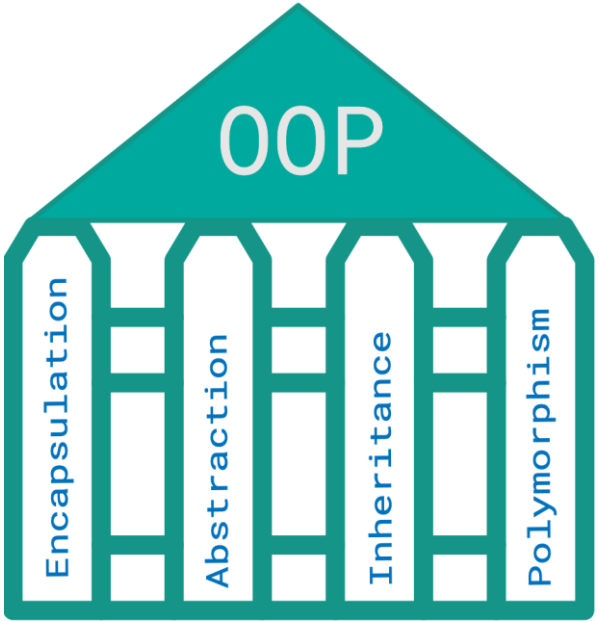


Procedural Programming Based Solution

Object Oriented Programming Based Solution

- Which of the above solutions is cleaner & readable?
- Can we test the code easily?
- Can we reuse some parts of code wherever required with ease?

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

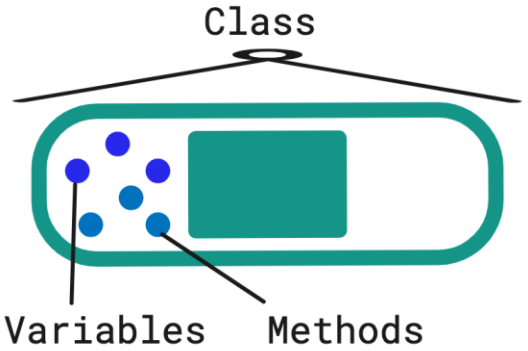
Can you figure out what could go wrong if this solution is used?

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.

# 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 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 Encapsulation?
  - 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 limits.
    - 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?

# 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);*
1. Let's fix the incrementTime() method

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?

# Take home exercises for the session

- You will explore encapsulation with this real world scenario in the following Byte
  - Encapsulation Byte - Crio.do
- Session 3 Java - 2 Quiz

All of these details are also available on the site.

# Feedback

Thank you for joining in today.

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

# Further Reading

- Bounding Box (Optional Assignment)

# References


- OOPs in Java: Encapsulation, Inheritance, Polymorphism, Abstraction (beginnersbook.com)
- Encapsulation – CS2 – Java (towson.edu)


# Thank you