BridgeLabz

Employability Delivered

Java Programming

Java Language

1. Java Terminology
2. Java is a Platform Independent Language
3. JVM Architecture
4. Java Features

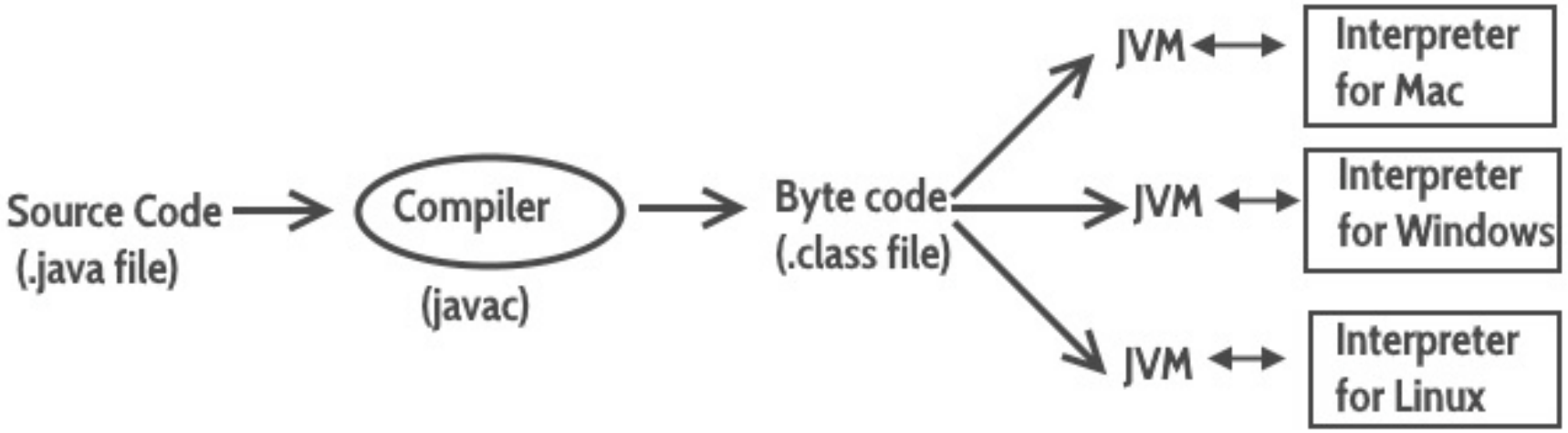
Java Terminology

- **Java Installation:** echo $(/usr/libexec/java_home)
- **Java Program Execution:** It starts with we writing the program, then compiling the program and at last running the program.
- **Java Virtual Machine (JVM):** The primary function of the JVM is execute the bytecode produced by compiler while running the program. Each operating system has different JVM, however the output they produce after execution of bytecode is same across all operating systems. That is why we call java as platform independent language.
- **Bytecode:** javac compiler of JDK compiles the java source code into bytecode so that it can be executed by JVM. The bytecode is saved in a .class file by compiler.
- **Java Development Kit(JDK):** JDK is platform specific installation and it includes JRE (Java Runtime Environment), compilers and various tools like JavaDoc, Java debugger etc.
- **Java Runtime Environment(JRE):** JRE is part of JDK and when JRE is only installed then only Java program can be run but not compiled. JRE includes JVM, browser plugins and applets support.

Java is a platform independent language

BridgeLabz
Employability Delivered

Compiler(javac) converts source code (.java file) to the generic byte code(.class file). The bytecode can be run in any platform using OS specific JRE. Each OS has different JVM, however the output they produce after execution of bytecode is same across all operating systems.



JVM Architecture

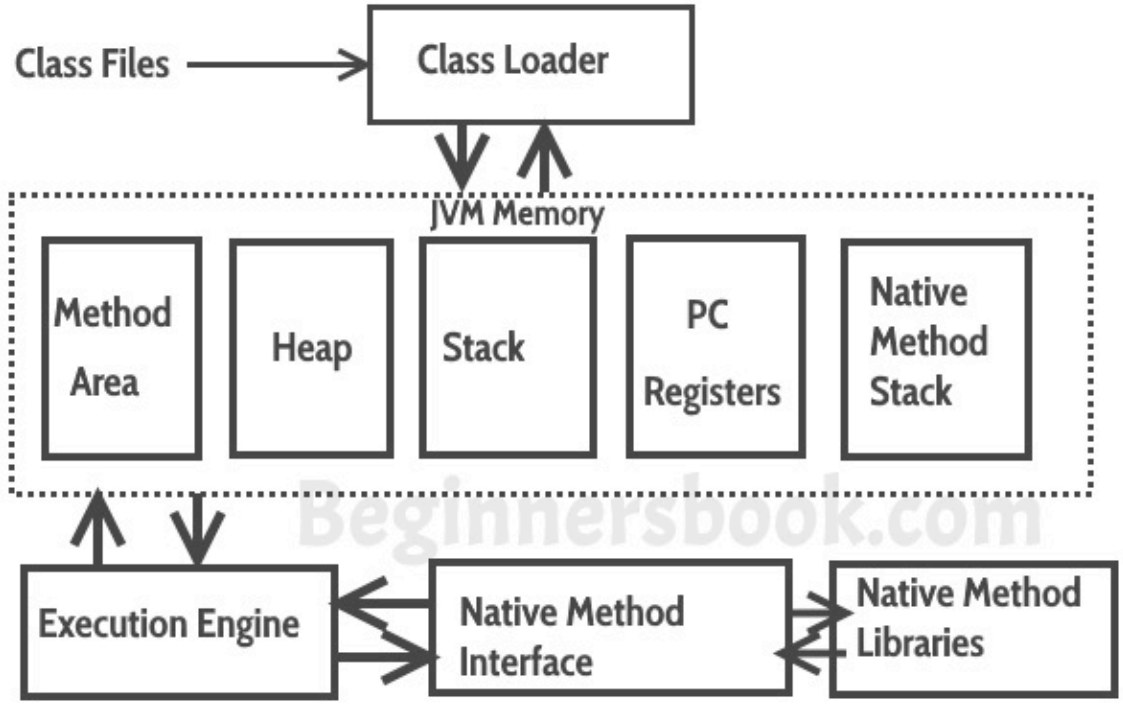
Class Loader: The class loader reads the .class file and save the byte code in the method area.

Method Area: There is only one method area in a JVM which is shared among all the classes. This holds the class level information of each .class file.

Heap: Heap is a part of JVM memory where objects are allocated. JVM creates a Class object for each .class file.

Stack: Stack is also a part of JVM memory but unlike Heap, it is used for storing temporary variables.

PC Registers: This keeps the track of which instruction has been executed and which one is going to be executed. Since instructions are executed by threads, each thread has a separate PC register.



Native Method stack: A native method can access the runtime data areas of the virtual machine.

Native Method interface: It enables java code to call or be called by native applications. Native applications are programs that are specific to the hardware and OS of a system.

Java Features

- **Java is an Object Oriented language:** Java is a pure Object oriented language where even writing the smallest of the Program starts with writing a class. Its essentially a way of organizing programs as collection of objects, each of which represents an instance of a class. And Classes are Organized using Object Oriented Concepts of Abstraction, Encapsulation, Inheritance, Polymorphism, and Association.
- **Robust Language:** Robust means reliable. Its made reliable by checking possible errors at compile time and runtime and supporting runtime features like Garbage Collection, Exception Handling and memory management which also makes it Secure.
- **Java is distributed:** The java programs can be distributed on more than one systems that are connected to each other using internet connection. Objects on one JVM (java virtual machine) can execute procedures on a remote JVM.
- **Multithreading:** Multithreading is a Java feature that allows concurrent execution of two or more parts of a program for maximum utilization of CPU.
- **Java Runtime Environment(JRE):** JRE is part of JDK and when JRE is only installed then only Java program can be run but not compiled. JRE includes JVM, browser plugins and applets support.

Java Programming Constructs

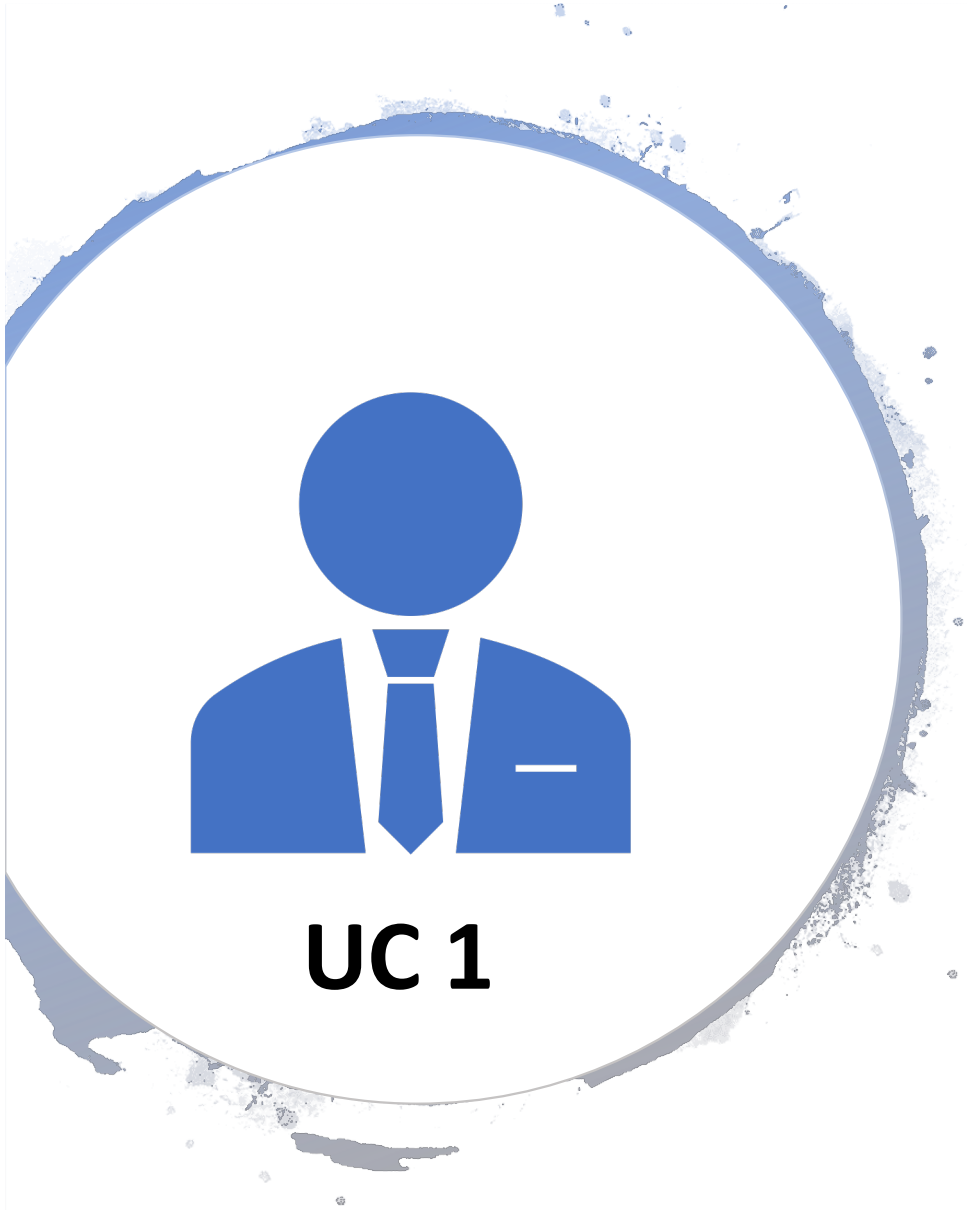
1. Sequences, Selection & Repetition
2. Class, Object and Methods
3. Key Concepts
4. Patterns

Note:  java cheatsheet

1. Sequences, Selection & Repetition

BridgeLabz
Employability Delivered

- Sequences are simple Java Statement
- A selection statement provides for selection between alternatives. Consists of if, else & switch statements
- A repetition construct causes a group of one or more program statements to be invoked repeatedly until some end condition is met. Consists of Fixed for loop and Variable while loop



Check Employee is Present or Absent

BridgeLabz
Employability Delivered

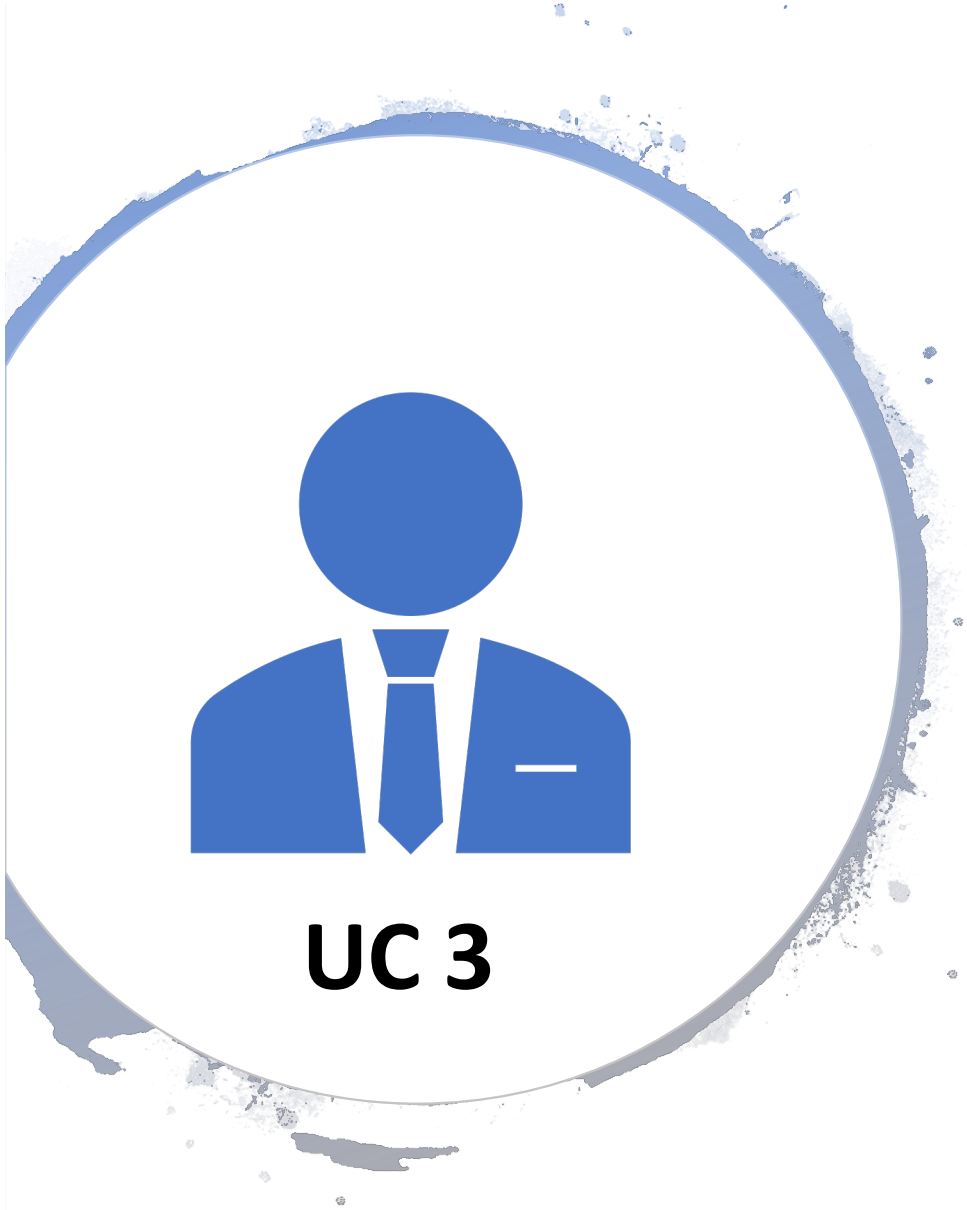
Check Employee Presence UC 1

```
public class EmpWageBuilderUC1 {

    public static void main(String[] args) {
        // Constants
        int IS_FULL_TIME = 1;
                // Computation
        double empCheck = Math.floor(Math.random() * 10) % 2;
        if (empCheck == IS_FULL_TIME)
            System.out.println("Employee is Present");
        else
            System.out.println("Employee is Abscent");
    }
}
EmpWageBuilderUC1.java (END)
```

Calculate Daily Employee Wage

Calculating Employee Wage UC 2

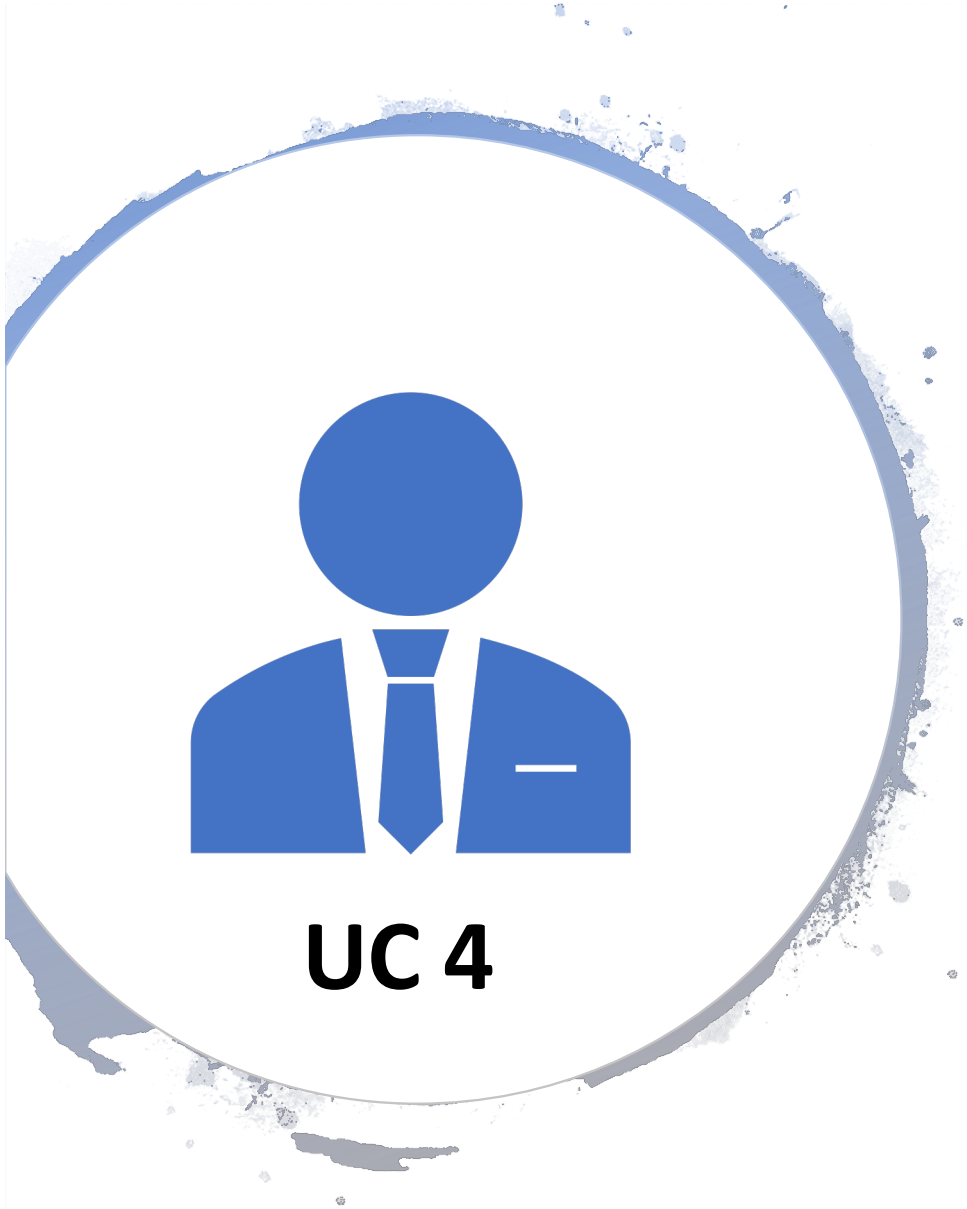
```
public class EmpWageBuilderUC2 {

    public static void main(String[] args) {
        // Constants
        int IS_FULL_TIME = 1;
        int EMP_RATE_PER_HOUR = 20;
        // Variables
        int empHrs = 0;
        int empWage = 0;
                  // Computation
        double empCheck = Math.floor(Math.random() * 10) % 2;
        if (empCheck == IS_FULL_TIME)
            empHrs = 8;
        else
            empHrs = 0;
        empWage = empHrs * EMP_RATE_PER_HOUR;
        System.out.println("Emp Wage: " + empWage);
    }
}
EmpWageBuilderUC2.java (END)
```



Add Part time Employee & Wage

Add Part time Employee

```
public class EmpWageBuilderIf {

    public static void main(String[] args) {
        // Constants
        int IS_PART_TIME = 1;
        int IS_FULL_TIME = 2;
        int EMP_RATE_PER_HOUR = 20;
        // Variables
        int empHrs = 0;
        int empWage = 0;
        // Computation
        double empCheck = Math.floor(Math.random() * 10) % 3
        if (empCheck == IS_PART_TIME)
            empHrs = 4;
        else if (empCheck == IS_FULL_TIME)
            empHrs = 8;
        else
            empHrs = 0;
        empWage = empHrs * EMP_RATE_PER_HOUR;
        System.out.println("Emp Wage: " + empWage);
    }
}
EmpWageBuilderIf.java (END)
```



Solving using Case Statement

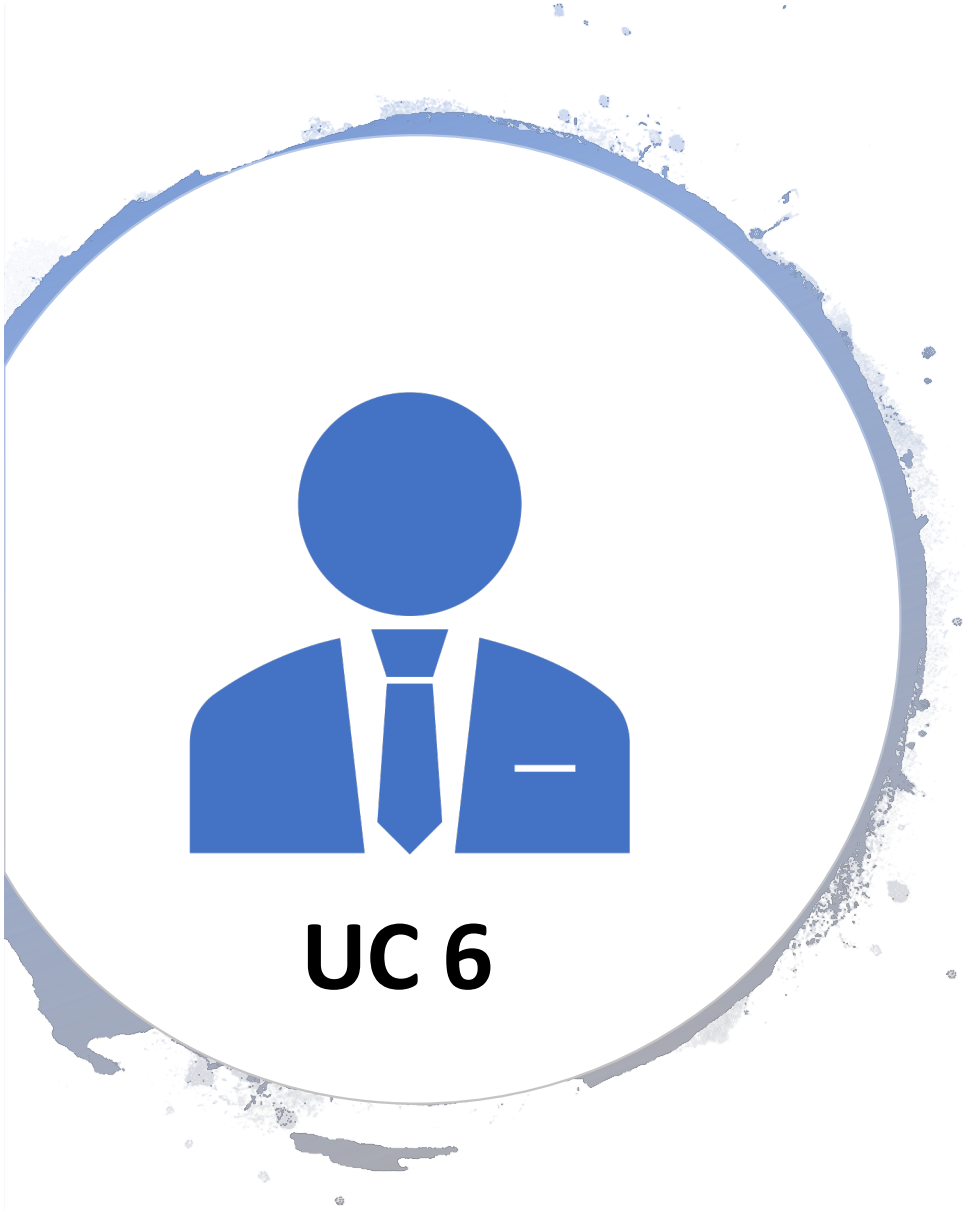
Calculating Employee Wage Using Switch

```
public class EmpWageBuilderCase {
    public static final int IS_PART_TIME = 1;
    public static final int IS_FULL_TIME = 2;
    public static final int EMP_RATE_PER_HOUR = 20;

    public static void main(String[] args) {
        // Variables
        int empHrs = 0;
        int empWage = 0;
        // Computation
        int empCheck = (int) Math.floor(Math.random() * 10) % 3;
        switch (empCheck) {
            case IS_PART_TIME:
                empHrs = 4;
                break;
            case IS_FULL_TIME:
                empHrs = 8;
                break;
            default:
                empHrs = 0;
        }
        empWage = empHrs * EMP_RATE_PER_HOUR;
        System.out.println("Emp Wage: " + empWage);
    }
}
EmpWageBuilderCase.java (END)
```

Calculating Wages for a Month assuming 20 Working Days in a Month

Calculating Wages for a Month

```
public class EmpWageBuilderFor {
    public static final int IS_PART_TIME = 1;
    public static final int IS_FULL_TIME = 2;
    public static final int EMP_RATE_PER_HOUR = 20;
    public static final int NUM_OF_WORKING_DAYS = 2;

    public static void main(String[] args) {
        // Variables
        int empHrs = 0, empWage = 0, totalEmpWage = 0;
        // Computation
        for (int day = 0; day < NUM_OF_WORKING_DAYS; day++) {
            int empCheck = (int) Math.floor(Math.random() * 10) % 3;
            switch (empCheck) {
                case IS_PART_TIME:
                    empHrs = 4;
                    break;
                case IS_FULL_TIME:
                    empHrs = 8;
                    break;
                default:
                    empHrs = 0;
            }
            empWage = empHrs * EMP_RATE_PER_HOUR;
            totalEmpWage += empWage;
            System.out.println("Emp Wage: " + empWage);
        }
        System.out.println("Total Emp Wage: " + totalEmpWage);
    }
}
EmpWageBuilderFor.java (END)
```



Calculate Wages till a condition of total working hours of 100 or max days os 20 is reached for a month

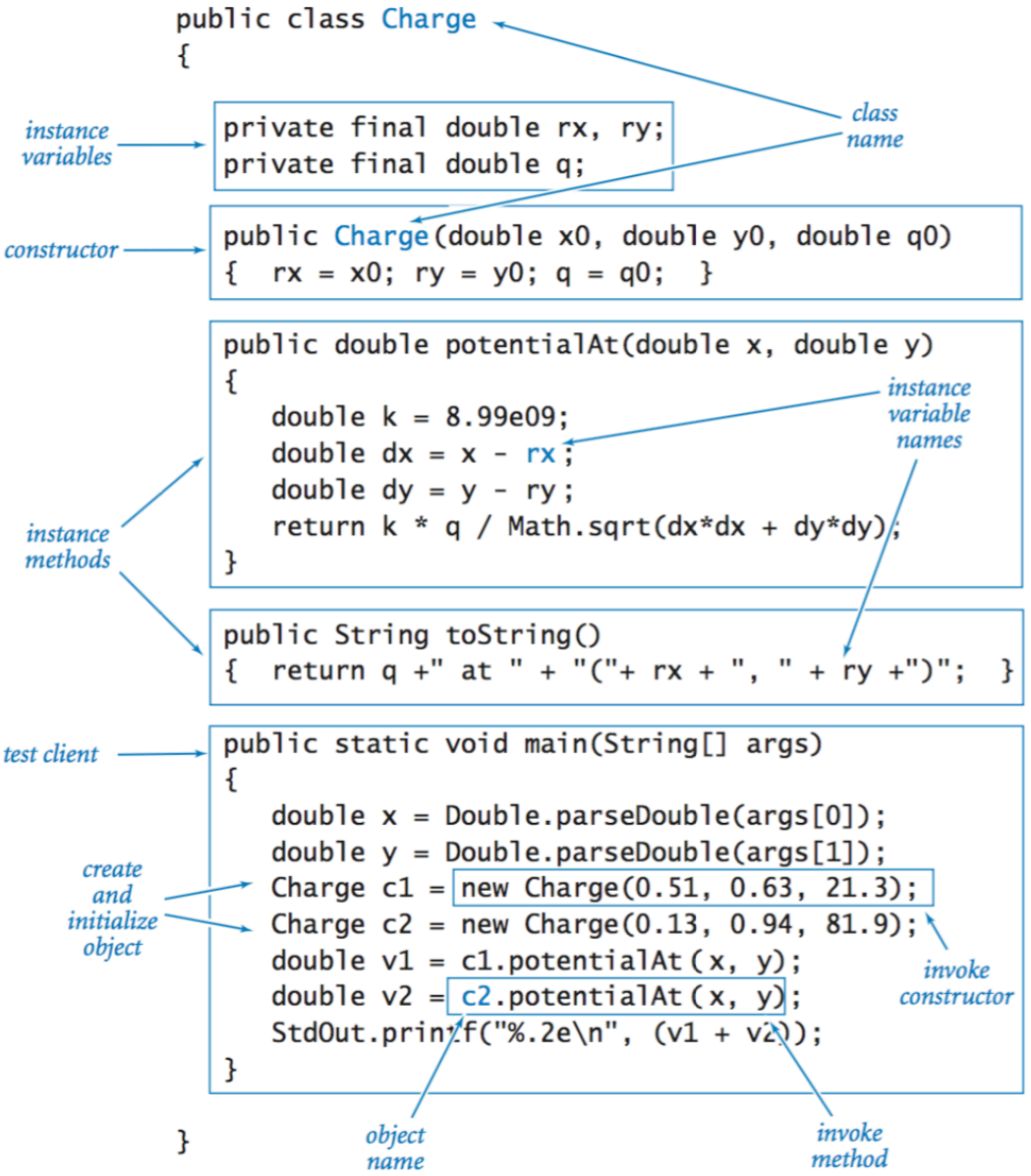
Calculating Wages till Number of Working Days or Total Working Hours per month is Reached

```
public class EmpWageBuilderWhile {
    public static final int IS_PART_TIME = 1;
    public static final int IS_FULL_TIME = 2;
    public static final int EMP_RATE_PER_HOUR = 20;
    public static final int NUM_OF_WORKING_DAYS = 2;
    public static final int MAX_HRS_IN_MONTH = 10;

    public static void main(String[] args) {
        // Variables
        int empHrs = 0, totalEmpHrs = 0, totalWorkingDays = 0;
        // Computation
        while (totalEmpHrs <= MAX_HRS_IN_MONTH &&
               totalWorkingDays < NUM_OF_WORKING_DAYS) {
            totalWorkingDays++;
            int empCheck = (int) Math.floor(Math.random() * 10) % 3;
            switch (empCheck) {
                case IS_PART_TIME:
                    empHrs = 4;
                    break;
                case IS_FULL_TIME:
                    empHrs = 8;
                    break;
                default:
                    empHrs = 0;
            }
            totalEmpHrs += empHrs;
            System.out.println("Day#: " + totalWorkingDays + " Emp Hr: " +empHrs);
        }
        int totalEmpWage = totalEmpHrs * EMP_RATE_PER_HOUR;
        System.out.println("Total Emp Wage: " + totalEmpWage);
    }
}
/Users/narayan/Development/CoreJava/temp/EmpWageBuilderWhile.java (END)
```

2. Class, Objects & Methods

- A **Class** can be considered as a blueprint using which you can create as many objects as you like.
- **Objects** have state and behaviour
- **Abstraction** is a process where you show only “relevant” data and “hide” unnecessary details of an object from the user.
- **Encapsulation** simply means binding object state(fields) and behaviour (methods) together. If you are creating class, you are doing encapsulation.
- **Association** establishes relationships between two Objects so as to enable Method Invocation

Class Specification



BridgeLabz
Employability Delivered

BridgeLabz

Employability Delivered

Thank You