Core Java

Day 03 Agenda

- Control statements
  - if statements
  - switch statements
  - loop statements
  - jump statements
  - ternary operators
- Java methods
- Class and Object
  - Fields
  - Getter/Setters
  - Facilitators
  - Constructors
- OOP concepts Major / Minor pillars

Control Statements

- By default, Java statements are executed sequentially i.e. statements are executed in order in which they appear in code.
- Java also provide statements to control the flow of execution. These are called as control statements.
- Types of control flow statements.
  - Decision Making statements
    - if statements
    - switch statement
  - Loop statements
    - do while loop
    - while loop
    - for loop
    - for-each loop
    - labeled loop
  - Jump statements
    - break statement
    - continue statement
    - return statement
- Being structured programming language, control flow statements (blocks) can be nested within each other.

if statements

- In Java, conditions are boolean expressions that evaluate to true or false.
- Program execution proceeds depending on condition (true or false).
- Syntax:

```
if(condition) {
    // execute when condition is true
}
```

```
if(condition) {
    // execute when condition is true
}
else {
    // execute when condition is false
}
```

```
if(condition1) {
    // execute when condition1 is true
}
else if(condition2) {
    // execute when condition2 is true
}
else {
    // execute when condition no condition is true
}
```

switch statements

- Selects a code block to be executed based on given expression.
- The expression can be integer, character or String type.

```
switch (expression) {
    case value1:
        // executed when expression equals value1
    break;
    case value2:
        // executed when expression equals value2
    break;
    // ...
    default:
        // executed when expression not equals any case constant
}
```

- We can use String constants/expressions for switch case in Java (along with integer and char types).

```
String course = "DAC";
switch(course) {
case "DAC":
    System.out.println("Welcome to DAC!");
```

```
        break;
    case "DMC":
        System.out.println("Welcome to DMC!");
        break;
    case "DESD":
        System.out.println("Welcome to DESD!");
        break;
    default:
        System.out.println("Welcome to C-DAC!");
    }
```

do-while loop

- Executes loop body and then evaluate the condition.
- If condition is true, loop is repeated again.

```
do {
    // body
} while(condition);
```

- Typically used to implement menu-driven program with swicth-case.

while loop

- Evaluate the condition and repeat the body if condition is true.

```
initialization;
while(condition) {
    body;
    modification;
}
```

for loop

- Initialize, evaluate the condition, execute the body if condition is true and then execute modification statement.

```
for(initialization; condition; modification) {
    body;
}
```

for-each loop

- Execute once for each element in array/collection.

```
int[] arr = { 11, 22, 33, 44 };
for(int i: arr)
    System.out.println(i);
```

break statement

- Stops switch/loop execution and jump to next statement after the switch/loop.

```
initialization;
while(condition) {
    body;
    if(stop-condition)
        break;
    modification;
}
statements after loop;
```

continue statement

- Jumps to next iteration of the loop.

```
initialization;
while(condition) {
    if(skip-condition)
        continue;
    body;
    modification;
}
statements after loop;
```

Labeled loops

- In case of nested loop, break and continue statements affect the loop in which they are placed.
- Labeled loops overcome this limitation. Programmer can choose the loop to be affected by break/continue statement.
- Labels can be used with while/for loop.

```
outer: for(int i=1; i<=3; i++) {
    middle: for(int j=1; j<=3; j++) {
        inner: for(int k=1; k<=3; k++) {
            if(i==j && j==k && i==K)
                break middle;
            System.out.printf("%d, %d, %d\n", i, j, k);
        }
    }
}
```

```
2 1 1
2 1 2
2 1 3
2 2 1
3 1 1
3 1 2
3 1 3
3 2 1
3 2 2
3 2 3
3 3 1
3 3 2
```

Ternary operator

- Ternary operator/Conditional operator

```
condition? expression1 : expression2;
```

- Equivalent if-else code

```
if(condition)
    expression1;
else
    expression2;
```

- If condition is true, expression1 is executed and if condition is false, expression2 is executed.

```
a = 10;
b = 7;
max = (a > b) ? a : b;
```

```
a = 10;
b = 17;
max = (a > b) ? a : b;
```

Java methods

- A method is a block of code (definition). Executes when it is called (method call).
- Method may take inputs known as parameters.
- Method may yield a output known as return value.
- Method is a logical set of instructions and can be called multiple times (reusability).

```
class ClassName {
    public static ret-type staticMethod(parameters) {
        // method body
    }

    public ret-type nonStaticMethod(parameters) {
        // method body
    }

    public static void main(String[] args) {
        // ...
        res1 = ClassName.staticMethod(arguments);
        ClassName obj = new ClassName();
        res2 = obj.nonStaticMethod(arguments);
    }
}
```

Class and Object

- Class is collection of logically related data members ("fields"/attributes/properties) and the member functions ("methods"/operations/messages) to operate on that data.
- A class is user defined data type. It is used to create one or more instances called as "Objects".
- Class is blueprint/prototype/template of the object; while Object is an instance of the class.
- Class is logical entity and Object represent physical real-world entity.
- Class represents group of such instances which is having common structure and common behavior.

- class is an imaginary entity.
  - Example: Car, Book, Laptop etc.
  - Class implementation represents encapsulation.
  - e.g. Human is a class and You are one of the object of the class.

```
class Human {
    int age;
    double weight;
    double height;
    // ...
    void walk() { ... }
    void talk() { ... }
    void think() { ... }
    // ...
}
```

- Since class is non-primitive/reference type in Java, its objects are always created on heap (using new operator). Object creation is also referred as "Instantiation" of the class.

  ```
  Human obj = new Human();
  obj.walk();
  ```

Instance

```
- Definition
    1. In java, object is also called as instance
    2. An entity, which is having physical existance is called as instance.
    3. An entity, which is having state, behavior and identity is called as
instance.
- instance is a real time entity.
- Example: "Tata Nano", "Java Complete Reference", "MacBook Air" etc.
```

- Types of methods in a Java class.
  - Methods are at class-level, not at object-level. In other words, same copy of class methods is used by all objects of the class.
  - Parameterless Constructor
    - In Java, fields have default values if unintialized. Primitive types default value is usually zero (refer data types table) and Reference type default value is null. Constructor should initialize fields to the desired values.
    - Has same name as of class name and no return type. Automatically called when object is created (with no arguments).
    - If no construcor defined in class, Java compiler provides a default construcor (Parameterless).

    ```
    Human obj = new Human();
    ```

  - Parameterized Constructor
    - Constructor should initialize fields to the given values.
    - Has same name as of class name and no return type. Automatically called when object is created (with arguments).

    ```
    Human obj = new Human(40, 76.5, 5.5);
    ```

  - Inspectors/Getters
    - Used to get value of the field outside the class.

    ```
    double ht = obj.getHeight();
    ```

- Mutators/Setters
  - Used to set value of the field from outside the class.
  - It modifies state of the object.

  ```
  obj.setHeight(5.5);
  ```

  - Getter/setters provide "controlled access" to the fields from outside the class.
- Facilitators
  - Provides additional functionalities like Console IO, File IO.

  ```
  obj.display();
  ```

- Other methdos/Business logic methods

  ```
  obj.talk();
  ```

- Executing a method on object is also interpreted as
  - Calling member function/method on object.
  - Invoking member function/method on object.
  - Doing operation on the object.
  - Passing message to the object.
- Each object has
  - State: Represents values of fields in the object.
  - Behaviour: Represents methods in the class and also depends on Object state.
  - Identity: Represents uniqueness of the object.
- Object created on heap using new operator is anonymous.

  ```
  new Human().talk();
  ```

- Assigning reference doesn't create new object.

  ```
  Human h1 = new Human(...);
  Human h2 = h1;
  ```

How to get system date in Java?

- Using Calender class:

```
Calendar c = Calendar.getInstance();
//int day = c.get( Calendar.DAY_OF_MONTH );
int day = c.get( Calendar.DATE );
int month = c.get( Calendar.MONTH) + 1;
int year = c.get( Calendar.YEAR );
```

Initialization

```
int num1 = 10;       //Initialization
int num2 = num1;     //Initialization
```

- Initialization is the process of storing value inside variable during its declaration.
- We can initialize any variable only once.

Assignment

```
int num1 = 10;  //Initialization
//int num1 = 20;  //Not OK
num1 = 20;  //OK:   Assignment
num1 = 30;  //OK:   Assignment
```

- Assignment is the process of storing value inside variable after its declaration.
- We can do assignment multiple times.

Constructor

- It is a method of class which is used to initialize instance.
- Constructor is a special syntax of Java because:
    1. Its name and class name is always same.
    2. It doesn't have any return type
    3. It is designed to call implicitly.
    4. In the lifetime on instance, it gets called only once.

```
public Date(  ){    //Constructor of the class
    System.out.println("Inside constructor");
    Calendar c = Calendar.getInstance( );
    this.day = c.get( Calendar.DATE );
    this.month = c.get( Calendar.MONTH ) + 1;
    this.year = c.get( Calendar.YEAR );
}
```

- We can not call constructor on instance explicitly.

```
Date date = new Date(); //OK
date.Date( );    //Not OK
```

- We can use any access modifer on constructor.
- If constructor is public then we can create instance of a class inside method of same class as well as different class.
- If constructor is private then we can create instance of class inside method of same class only.
- Types of constructor in Java:
    1. Parameterless constructor.
    2. Parameterized constructor.
    3. Default constructor.

Parameterless constructor

- A constructor which do not have any parameter is called parameterless constructor.

```
    public Date(  ){   //Parameterless constructor
        Calendar c = Calendar.getInstance( );
        this.day = c.get( Calendar.DATE );
        this.month = c.get( Calendar.MONTH ) + 1;
        this.year = c.get( Calendar.YEAR );
    }
```

- If we create instance W/O passing arguments then parameterless constructor gets called.

```
Date dt1 = new Date( );  //OK
```

- In the above code, parameterless constructor will call.

Parameterized constructor

- Constructor of a class which take parameters is called parameterized constructor.

```
public Date( int day, int month, int year ){    //Parameterized constructor
    this.day = day;
    this.month = month;
    this.year = year;
}
```

- If we create instance by passing arguments then parameterized constructor gets called.

```
Date date = new Date( 23, 7, 1983 );    //OK
```

- Constructor calling sequence depends on order of instance creation.

Default constructor

- If we do not define any constructor(no parameterless, no parameterized) inside class then compiler generate one constructor for the class by default. It is called default constructor.
- Compiler generated default constructor is zero parameter i.e parameterless constructor.
- Compiler never generate default parameterized constructor. It means that, if we want to create instance with arguments then we must define parameterized constructor inside class.

Constructor chaining

- In Java, we can call constructor from another constructor. It is called constructor chaining.
- For constructor chaining, we should use this statement.
- this statement must be first statement inside constructor body.

```java
class Date{
    private int day;    //Default value is 0
    private int month;  //Default value is 0
    private int year;   //Default value is 0

    public Date(  ){    //Parameterless Constructor
        this( 12, 8, 2022);    //Constructor chaining
    }
    public Date( int day, int month, int year ){    //Parameterized constructor
        this.day = day;
        this.month = month;
        this.year = year;
    }
}
```

- Using constructor chaning, we can reduce developers efforts.

this reference

- If we call non static method on instance then non static method get this reference.
- this reference contains reference of current/calling instance.
- Using this reference, non static method and non static field can communicate with each other. Hence this reference is considered as a link/connection between them.
- this reference is considered as a first parameter to the method. Hence it gets space once per method call on Java Stacks.
- We can not use this reference to access local variable. We should use this reference to access non static field/method.
- Use of this reference to access non static field/method is optional.
- If name of the local variable and name of field is same then to refer field we should use this reference.

```java
class Complex{
    private int real;
    public void setReal( int real ){
        this.real = real;
    }
}
```

- Definition:
    - this reference is implicit parameter available inside every non static method of the class which is used to store reference of current/calling instance.

OOPS concepts

- Following members do not get space inside instance
    1. Method parameter(e.g this reference, args etc)
    2. Method local variable
    3. Static field
    4. Static as well as non static method
    5. Constructor
- Only non static field get space inside instance.
- If we declare non static field inside class then it gets space once per instance according their order of declaration.

```
class Test{
    private int num1;
    private int num2;
    private int num3;
}
class Program{
    public static void main(String[] args) {
        Test t1 = new Test( );
        Test t2 = new Test( );
        Test t3 = new Test( );
    }
}
```

- Method do not get space inside instance. All the instances of same class share single copy of method declared inside class.
- Instances can share method by passing reference of the instance as argument to the method.

Object Oriented Programming Structure / System(OOPS)

- OOPS is not a syntax. Rather it is a process or a programming methodology that we can use to solve real world use cases / problems. In other words, oops is an object oriented thought process.
- Alan Kay is inventor of oops.
- Ref: https://medium.com/javascript-scene/the-forgotten-history-of-oop-88d71b9b2d9f
- Object Oriented Analysis and Design with application: Grady Booch
- According to Grady Booch, there are 4 major and 3 minor pillars of oops.

4 Major pillars / parts / elemets

1. Abstraction : To achive simplicity
2. Encapsulation : To achive data hiding and data security
3. Modularity : To minimize module dependency
4. Hierarchy : To achive reusability

- By major, we mean that a language without any one of these elements is not object oriented.

3 Minor pillars / parts / elemets

1. Typing : To minimize maintenance of the system.
2. Concurrency : To utilize hardware resources efficiently.
3. Persistence : To maintain state of the instance on secondary storage

- By minor, we mean that each of these elements is a useful, but not essential.

Abstraction

- It is a major pillar of oops.
- Abstraction is the process of getting essential things from system/object.
- Abstraction focuses on the essential characteristics of som object, relative to the perspective of viewer. In simple word, abstraction changes from user to user.
- Abstraction describes external behavior of an instance.
- Creating instance and calling methods on it represents abstraction programatically.

```java
Scanner sc = new Scanner( System.in );
String name = sc.nextLine( );
int empid = sc.nextInt( );
float salary = sc.nextFloat( );
```

```java
Complex c1 = new Complex();
c1.acceptRecord( );
c1.printRecord( );
```

- Using abstraction, we can achive simplicity.

Encapsulation

- It is a major pillar of oops.
- Definition:
    1. To achive abstraction, we need to provide some implementation. This implementation of abstraction is called encapsulation.
    2. Binding of data and code together is called as encapsulation.
- Hiding represents encapsulation.

```
class Complex{
    /* Data*/
    private int real;
    private int imag;
    public Complex( ){
        this.real = 0;
        this.imag = 0;
    }
    /*Code*/
    public void acceptRecord( ){
        Scanner sc = new Scanner( System.in)
        //TODO: accept record for real and imag
    }
    public void printRecord( ){
        //TODO: print state of real and imag
    }
}
class Program{
    public static void main(String[] args) {
        //Abstraction
        Complex c1 = new Complex();
        c1.acceptRecord( );
        c1.printRecord( );
    }
}
```

- Abstraction and encapsulation are complementary concepts: Abstraction focuses on the observable behavior of an object, whereas encapsulation focuses on the implementation that gives rise to this behavior.
- Class implementation represents encapsulation.
- Use of encapsulation:
    1. To achive abstraction
    2. To achive data hiding(also called as data encapsulation).
- Process of declaring field private is called as data hiding.
- Data hiding help to achive data security.

Modularity

- It is a major pillar of oops
- Modularity is the process of designing and developing complex system using small parts/modules.
- Main purpose of modularity is to minimize module dependancy.
- Using .jar file, we can achive modularity in Java.

Hierarchy

- It is a major pillar of oops.
- Level/order/ranking of abstraction is called as hierarchy.
- Using hierarchy, we can achive code reusability.
- Application of reusability:

    1. To reduce development time.
    2. To reduce development cost
    3. To reduce develpers effort.
- Types of hierarchy:
    1. Has-a => Association
    2. Is-a => Generalization(is also called as inheritance)
    3. Use-a => Dependancy
    4. Create-a => Instantiation

Typing

- It is a minor pillar of oops.
- It is also called as polymorphism(poly(many) + morphism(forms/behavior)).
- polymorphism is a Greek word.
- An ability of any instance to take multiple forms is called as polymorphism.
- Using typing/polymorphism, we can reduce maintenance of the application.
- In Java, we can achive polymorphism using two ways:
    1. Method overloading (It represents compile time polymorphism)
    2. Method overriding (It represents run time polymorphism)
- In Java, runtime polymorphism is also called as dynamic method dispatch

Concurrency

- It is a minor pillar of oops.
- Concurrency is the process of executing multiple task simultaneously.
- Using thread, we can achive Concurrency in Java.
- Using Concurrency, we can utilize H/W resources efficiently.

Persistance

- It is a minor pillar of oops.
- Process of maintaining state of instance inside file / database is called as Persistance.
- We can implement it using file handing(serialization / derserialization) and database programming(JDBC).

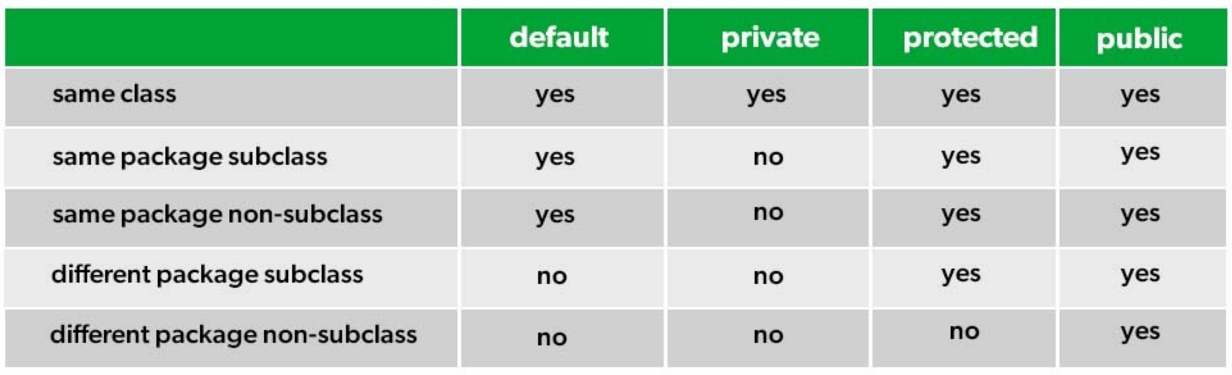
Access modifiers (for types)

- default: When no specifier is mentioned.
    - The types are accessible in current package only.
- public: When "public" specifier is mentioned.
    - The types are accessible in current package as well as outside the package (using import).

Access modifiers (for type members)

- private: When "private" specifier is mentioned before the member field/method.
    - The members are accessible in current class (member OR this.member).
- default: When no specifier is mentioned before the member field/method.
    - The members are accessible in current class (member OR this.member).

  - The members are accessible in all classes in same package (obj.member OR ClassName.member).
- protected: When "protected" specifier is mentioned before the member field/method.
  - The members are accessible in current class (member OR this.member).
  - The members are accessible in all classes in same package including its sub-classes (super.member, obj.member OR ClassName.member).
  - The members are accessible in sub classes outside the package including its sub-classes (super.member).
- public: When "public" specifier is mentioned before the member field/method.
  - The members are accessible in current class (member OR this.member).
  - The members are accessible in all other classes (super.member, obj.member OR ClassName.member).
- Scopes
  - private (lowest)
  - default
  - protected
  - public (highest)



	default	private	protected	public
same class	yes	yes	yes	yes
same package subclass	yes	no	yes	yes
same package non-subclass	yes	no	yes	yes
different package subclass	no	no	yes	yes
different package non-subclass	no	no	no	yes

-

Assignments

1. Input a month and year from the user. Print number of days in the month. Note that in leap year, February has 29 days.
2. Print fibonacci series. Take number of terms from the user. Fibonacci series is 0, 1, 1, 2, 3, 5, 8, ... Here each term is addition of two previous terms.
3. Write a program that inputs employee details like emp number, name, department, job title, salary, hra (house rent), da (dearness) from the user. Program should print these details back along with gross salary.
4. Write a class Human as shown in Notes. Write System.out.println() statements in talk(), walk() and think() methods to show appropriate messages. Implement constructors, getter/setters, accept() and display() methods.
5. Write a class Box with fields length, breadth, and height. Implement constructors, getter/setters, accept() and display() methods. Also provide methods that calculate volume and surface area and reurn from the method.

6. Write a class Date with fields day, month, and year. Implement constructors, getter/setters, accept() and display() methods.
7. Write a class Fraction with fields numerator and denominator. Implement constructors, getter/setters, accept() and display() methods. Note that denominator cannot be zero. Write following logic methods in the class.
    - double decimalValue(); // returns decimal value of the fraction
    - boolean isSame(Fraction that);// returns true if "this" fraction is same as "that" fraction.
    - boolean isProper(); // returns true if numerator < denominator.