Core Java

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

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.

```
while(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.
    - 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();
```

- 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.

```
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.

Object oriented programming

1. Understand problem statement and try to solve it using OO concepts(class and instance)
2. Decide class and declare fields inside.
    - Variable declared inside class is called as field.
    - Non static field declared inside class is called instance variable.
    - In java, object is also called as instance.
    - Instance variable get space inside instance.
    - Instance variable get space once per instance according to their order of declaration inside class.
3. Create instance of a class.
    - new is an operator in java. It is used to allocate memory for instance of Heap.
    - Process of creating instance from a class is called instantiation
    - Only instance variable get space inside instance.
    - An instance without name is called anonymous instance.
    - Value stored inside instance is called state of the instance.
    - If we want to peform operation on instance then we should create reference of the instance.
    - Non static field i.e instance variable get space once per instance.
    - instance variable get space according to their order of declaration inside class.

```
//Date birthDate; //object reference / reference
//new Date( );    //Anonymous instance

//Date birthDate; //object reference / reference
//birthDate = new Date( );    //Anonymous instance

Date birthDate = new Date( );   //Instance with reference
```

```
- To access instance variable we should use reference.
```

4. If we want to access state(also called as value) of the instance then we should perform operation on that instance. In other words, we should call method on instance.
    - Process of calling method on instance is called message passing.

Access Modifier

```
- If we want to control visibility of the members of the class then we should use
access modifiers.
- There are 4 access modifiers in Java:
    1. private
    2. package level private( also called as default )
    3. protected
    4. public
```

this reference

```
- If we want to access/modify state( also called as value ) of the instance then
we should call method on instance.
- If we call method on instance then compiler implicitly pass reference of the
instance as a argument to that method.
- To store reference, compiler implicitly declare one parameter inside method.
Such parameter is called this reference.
- this is a keyword in Java.
- We can not pass / declare this reference explicitly.
- this reference is a local variable hence it will get space once per method on
Java stack.
- Using this reference, non static fields and non static methods can communicate
with each other. Hence this reference is also called as link or connection between
them.
- Definition: this reference is implicit reference variable which is available
inside every non static method of the class and which is used to store reference
of current instance.
- Inside method, to access members of the class, use of this keyword is optional.
```

```java
static int sum( int x, int y ){ //x and y   => parameters / Method parameters
    //TODO
}
public static void main( String[] args ){
    int a = 10;
    int b = 20;
    sum( a, b );    //a and b   => Arguments / Method arguments
}
```

- A function implemented or defined inside the class called as method

```
class Program
{
    //non-static method => Instance method
    public void print(){
           //TODO
    }
    // static method => Class level method
    public static void main(String[ ] args){
           //TODO
    }
}
```

- variable declared outside the method / inside the class is called field

```
class Program
{
    //field / property
    private int num1; // OK  // non-static field => instance variable
    private static int num1; // OK  // Static field => Class level variable

    //C/C++ function
    public static void main(String[ ] args){ // method / operation / behaviour /
message

    //datatype variable-name
    int num3; // OK // non-static local variable // Method local variable
    static int num4; // not ok  (In java local variable not static )


    }
}
```

```
class A  //Concrete class
{
    public int num1; // Non static - Field (it is also called as instance
variable)

    // Concrete method
    public void print( ){   // Non static -Method ( it is also called as instance
method )
        //TODO
    }
}
class Program
{
    public static void main( String args[ ]){
        //num1 = 10; // Compiler error (Belongs to class A )
```

```
        //print( ); // Compiler error

        //A a; // a is reference  / object reference
        //new A( ); // Instance of class A
        A a = new A( ); //Instance with reference
        a.num1 = 10; // OK
        a.print( ); // OK
    }
}
//Instance members = instance variable + instance method
```

- A class from which we can create instance is called concrete class
- To create instance we should use new operator
- If we want to access non-static members / instance members then we should use object instance
- A method of a class which is having a body is called as concrete method

```
class B
{
    public static int num1; // Static Field(it is called as class  level variable)
    //Concrete method
    public static void print(){ // Static Method(it is called as class level
method)

    }
}
class Program
{
    public static void main( String args[ ]){
        B.num1 = 10; // OK
        B.print( ); // OK
    }
}
//Class level members = class level variable + class level method
//Instance members = instance variable + instance method
```

- To access static members / class level members we should use class name and dot operator.

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.