BASICS

**HOW JAVA RUNS?**

1) JDK - full package kit, contains compiler, debugger,JVM, JRE to build java applications
2) JRE - contains only JVM, libraries, java.util not compile ,if you want to execute the .class files this is enough .
3) Under JVM - class loader - loads your .class files to the memory,**Bytecode Verifier**: checks that the code is safe and valid (no hacking attempts, memory violations). **Interpreter**: reads bytecode **line-by-line** and **executes** it. **JIT Compiler** (Just-In-Time compiler): to make faster execution.

You (write .java file)
↓
IDE (calls javac from JDK)
↓
Compiler (javac → creates .class file)
↓
JVM (inside JRE loads .class)
↓
ClassLoader → Bytecode Verifier → Interpreter + JIT
↓
Program runs on your computer!

**STARTING POINT TO EXECUTION**
Public static void main(String[] args) - command line arguments void and main cant be interchanged, - if you want to give inputs through command prompts - > eg) port info ,
Or
Static Public void main(String… args) - vargs

**KEYPOINTS**
Main methods can be overloaded - same method name with diff args
Cannot be overridden - because static methods cannot be override, does not deal with objects
Method hiding can be done

## DATA TYPES

JAVA is a strictly typed programming language.

| Primitive data types | NOn- PRIMITIVE DATA TYPES |
|---|---|
| Char, int ,long,short,byte double,float boolean | STring, Array, Our own defined class, collections |
| Pre- defined types and fixed size | Dynamic, depends on object |
| Not objects | These are objects |
| | |

**MEMORY**

Stack and heap memory
Local variables, method calls are stored in the stack . Heap - object , instance variables are stored;

**METHOD OVERLOADING /COMPILE TIME POLYMORPHISM**

Same method name with different parameters, diff order of parameters, and no of parameters ,no role with return .

**ENCAPSULATION**

Basically it means binding data and methods together. We should not allow access to the fields directly, outside the class. Thats why its defined as private adn access is done through getters and setters

**CONSTRUCTOR**

**Is a** special type of setter method whose name is same of the class name;
Invoked when an object is created . Used for initialising instance variables.
No return type, no void

```
Class Employee{

Employee(){

}
}
```

Default vs parameterised constructor
JVM creates the default constructor even if it is not mentioned explicitly.

| super() | this() |
|---|---|
| First line inside the constructor .it presents even if it is not mentioned explicitly. | First line inside the constructor.To call another constructor defined inside the class |
| | |

**Static**

Doesn’t deal with objects.

| Static block | Java initialization block |
|---|---|
| static<br>{<br>} | {<br>}  (has only curly braces) |
| This is called when the class gets loaded to the memory.and call only once. | This is called every time an object is created . |
| Understandable -  called before constructor before starting executing your program . | Called before constructor . |
| Used to initialise static variables or you want to do anything before running your program. | Used to initialise instance variables. |
| Multiple static blocks possible . in order of how you written top to bottom | Multiple initialization blocks possible . in order of how you written top to bottom |
| Before Main<br><br>If you want some code to run **even before the main`()` method starts**.<br><br>`Security check, license check, etc.DB connection set up` | Then what's the purpose betw  constructor and<br>Java initialization block  . if suppose have multiple constructors who share common data then we can add that in initialisation.Boilerplate code. |

Static variables are stored in the method area inside the JVM, not stack or heap.JVM has memory with 5 areas: stack, heap, method area.  Etc  so that static variables are shared among objects .Static variables **can be changed unless it says final.**

**STATIC METHODS CANNOT BE OVERRIDDEN**

**Becoz static deals with class not objects**

```
class Parent {
    static void display() {
        System.out.println("Parent Display");
    }
}

class Child extends Parent {
    static void display() {  // This is NOT overriding, it's called "method hiding"
        System.out.println("Child Display");
    }
}

public class Main {
    public static void main(String[] args) {
        Parent p = new Child();
        p.display();   // Output: Parent Display
    }
}
```

**Notice:**
**Even though the object is of `Child`,**
**since `display()` is static, it belongs to the class, not to the object.**

**Thus, no real overriding, only method hiding happens.**

**INHERITANCE**

Class which inherits the properties and methods of another class.
A class which inherits a child, sub class. A class which was inherited by someone is called base, parent,
Extends keyword
Specialised methods cannot be inherited if you have a reference variable that has parent , inherited, overridden .

CONSTRUCTOR cannot be inherited bcoz we force subclass to initialise the parent class instance variables.Each class should have its own way to initialize its variables.

**ACCESS MODIFIERS**

| **PUBLIC** | **PRIVATE** | **PROTECTED** | **DEFAULT** |
|---|---|---|---|
| **Can access everywhere** | **Access only within class** | **Can be accessed within class, folder with different class, and outside folder with a is a relationship concept . no to outside package with no relation.** | **Default methods within class and within folder diff class. But not access in the outside package .** |
| | | | |
| | | | |

**POLYMORPHISM**

**Many forms . in which the parent class reference refers to multiple child classes . And depending on the object type at runtime , the correct method is chosen .**

compile time polymorphism                    runtime polymorphism.

| Method overloading | Method overriding |
|---|---|
| Same method signature with different parameters, order | Redefining the parent method in child class with different behaviour. |

**ABSTRACTION**

Hiding implementation having only method signature.

**Abstract Class**

A class with a keyword abstract is enough to call a class as an abstract .
If a class has an abstract method then the class must be declared as an abstract.
It can have a constructor.
Doesn't have any implementation just tell you what to do not how it has to be done.
Concrete methods will be there.
Abstract variables are not allowed.
**Abstract methods should not be private because the class that inherits must implement those methods. Abstract cannot be applied to constructors.**

**Final keyword**

Apply on class - cannot be inherited
Apply on variables - cannot be changed
Apply on methods - cannot be overridden but involve inheritance.

**Interface**

Other way of achieving abstraction
By default, the methods defined under interface are **public and abstract.**
Doesn't have any implementation just tell you what to do not how it has to be done.
By defaults the variable under **interface are public static and final**
**Does not have a constructor because by default the variables are public static and final.**
**After java8, default methods and static methods, private methods are possible.**

| **Abstract class** | **Interface** |
|---|---|
| **Explicitly mention methods** | **By default methods are public and abstract** |
| **Can have constructor , to initialise variables** | **Cannot have constructor , to initialise variables** |
| **We can have protected except private** | **Methods access specifier is only public** |
| **Instance variables can be of any specifier** | **default the variables are public static and final.** |
| **abstract class indirectly participates in object creation** | **interface does not at all.** |
| **Static methods defined inside Abstract class can be inherited in subclass** | **Static methods defined inside Abstract class cannot be inherited in subclass** |

**Marker Interface**

Interface with no methods and fields . Purpose to give info to JVM that the class should be treated specially and implements marker interface . eg) fragile sticker on box - doesn't have anything except the sticker but they treat cautiously.

SErializable , clonable,remote

Custom creation

```
Interface auditable{
}


Class Animal implements auditable{

}
```

You can use instanceof to check or do operation only for those objects that implement the marker interface .

**JAVA 8 features**

Lambda expression
Streams
Default and static  methods in interface
Balanced tree implementation in hashing once it reaches the threshold
Functional interface
Optional class