JAVA CORE

-- JDK (Java Developement Kit)
JRE + Development tools(Compiler, Interpreter, Debugger, javadoc, keytool)
It has both developemnt tools and JRE.
If we want to develop and run the java programs then JDK is must

-- JRE (Java Runtime Environment)
JVM + Class Libraries or runtime libraries
JRE is used only to run Java programs
If we dont want to develop, but only to run the Java programs or Java applications, then JRE is enough
JRE is platform dependent.(Ex: I developed a java appli in windows and if I want to run it in linux or mac, it should have
JRE specific to linux or JRE specicfc to mac)

-- JVM (Java Virtual Machine)
It is the heart of Java Programming
It converts Java bytecode into machine specific code.
Platform dependent, provides garbage collectionWY, security, memory management features etc.
We can also customise JVM (allocation min and max memory to JVM)]
JVM is called Abstract virtual machine because, it provides an interface which does not depend on underlying OS or hardware.

-- Steps to Run Java Programs

	java compiler		JVM(JIT)		CPU	
Java Program	--->	Java Bytecode	--->	Machine code	--->	output
(helloWorld.java)		(helloworld.class)				

The java program is compiled by the Compiler present in JDK and generates Byte code with .class extension.
The bytecode is platform independent and runs on any device having the JRE.
After compilation, the bycode enters into Runtime process which has below steps
a) Class Laoder : The class loader dynamically loads the classes essential for running the program in the Java Virtual Machine.
When the JVM is started, three class loaders are used:
Bootstrap class loader: loads classes from jre/lib/rt.jar

Extensions class loader: loads classes from jre/lib/ext
System class loader: Loads classes from CLASSPATH
b) Byte code verifier
c) Interpreter

-- Difference between JDK, JRE, JVM
1. JDK is for development purpose, JRE is for running the java programs.
2. JDK, JRE both contain JVM, so we can run our java program.
3. JVM is the heart of the java programming , and provides platform independence.

-- JIT (Just in time Compiler) Fast compiler
JAVA uses the Just-in-time (JIT) interpreter along with the compiler for the program execution.
It is a part of JVM
It helps JVM to convert bytecode to machine code at run time
It performs some optimization on code.
Very frequently used methods are compiled once and kept into memory (cached).

— How can we say Java is platform independent?
The source codes gets converted to .class file which is in byte code by the compiler.
This byte code is platform independent, it can be run on any machine which has its specific JVM and JRE present in it (Jim, jre are platform dependent)
Java language is platform independent, whereas java software is platform dependent because different operating system have different JVMs
---

JAVA VARIABLES:
A variable is a container which holds the value while the Java program is executed
There are three types of variables in Java:

local variable : declared inside the body of the method. Cannot be declared as static
instance variable : declared inside the class but outside the body of the method. not declared as static.
static variable : A variable which is declared as static is called static variable. It cannot be local.
You can create a single copy of static variable and share among all the instances of the class.

Memory allocation for static variable happens only once (in MetaSpace memory) when the class is loaded in the memory.

MEMORY ALLOCATION:

1. STACK Memory:

Local Variables or Stack variables and Method invocations(calls) live on the STACK memory.

But if there is any object reference variable (instance variable) inside a method, it gets allocated on the HEAP memory

If Stack is full, java program throws StackOverflowError

2. HEAP Memory:OR Garbage Collectible Heap area

new generation heap

old generation heap

If Heap is full, the program throws OutofMemoryException

All Instance variables live on the HEAP, even inside a method body, it gets allocated on the HEAP not on the STACK

All object references get memory allocated in heap

3. MetaSpace: introduced in java 8

Classes, static variables are saved on this memory

4. String constant pool : String references are allocated here.

Each time you create a string literal, the JVM check the string constant pool. if the string already exist in the pool

a reference to the pool instance is returned. if the string does not exist in the pool, new string instance is created and placed

in the pool

Note: why java uses the concept of String literal|?

- to make java memory efficient (because no new object is created if it exist already in the string constant pool)

DATA TYPES:

There are two types of data types in Java:

1. Primitive data types: There are 8 primitive data types include boolean, char, byte, short, int, long, float and double.

2. Non-primitive data types: The non-primitive data types include Classes, Interfaces, and Arrays.

Data Type	Default Value	Default size	VALUE RANGE
boolean	false	1 bit	
char	'\u0000'	2 byte (16 bits)	(unicode) Its value-range lies between '\u0000' (or 0) to '\uffff' (or 65,535 inclusive).
byte	0	1 byte (8 bits)	-128 to 127
short	0	2 byte (16 bits)	-32,768 to 32,767
int	0	4 byte (32 bits)	(Whole numbers) - 2,147,483,648 (-2^31) to 2,147,483,647 (2^31 -1) .
long	0L	8 byte (64 bits)	(real numbers) -9,223,372,036,854,775,808(-2^63) to 9,223,372,036,854,775,807(2^63 -1)
float	0.0f	4 byte (32 bits)	(Single precision) The float data type is a single-precision 32-bit IEEE 754 floating point.Its value range is unlimited.
double	0.0d	8 byte (64 bits)	(double precision) The double data type is a double-precision 64-bit IEEE 754 floating point. Its value range is unlimited.

-- ACCESS MODIFIERS

There are two types of modifiers in Java: access modifiers and non-access modifiers.

ACCESS MODIFIERS: private, protected, public,default

NON ACCESS MODIFIERS:static, final, abstract,synchronized, native, volatile, transient etc.

-- Understanding Java Access Modifiers

Access Modifier	Within class	Within package	Outside package by subclass only	Outside package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

Note:Here default is more restrictive than protected. If you don't use any modifier, it is treated as default by default

---

-- TYPE CASTING

- Typecasting
Convert a value from one data type to another data type is known as type casting. The automatic conversion is done by the compiler and manual conversion performed by the programmer.
For example:
//here in the brackets we have mentioned long keyword, this is casting
double num = 10001.99;
long num2 = (long)num;
Types:
1. Widening Casting (automatically) - converting a smaller type to a larger type size.
byte -> short -> char -> int -> long -> float -> double.
Ex: int x = 10;
long y = x ; //automatically converts the integer type into long type
float z = y; //automatically converts the long type into float type

2. Narrowing Casting (manually) - converting a larger type to a smaller size type.
double -> float -> long -> int -> char -> short -> byte.
Ex: double d = 123.66;
long l = (long)d; //converting double data type into long data type
int i = (int)l; //converting long data type into int data type

output: d = 123.66
l = 123
i = 123

--UPCASTING and DOWNCASTING
Upcasting:
If the reference variable of Parent class refers to the object of Child class, it is known as upcasting.
For upcasting, we can use the reference variable of parent class type or an interface type
class A{}
class B extends A{}
A a=new B();//upcasting

DownCasting:
When Subclass reference variable refers to the object of Parent class, it is known as downcasting.

1. If we perform downcasting by typecasting, ClassCastException is thrown at runtime.
Ex: NexonCar nexoncar = (NexonCar)new Car; // Compiles successfully but ClassCastException is thrown at runtime
2. We can perform downcasting with or without instanceof operator

Usage:when we want to access specific behaviors of a subtype.
Here in the below example, if we want to get the subclass specific method (speed()) ,
we check if there is an instance of a NexonCar object passed in, downcast it to the NexonCar type and invoke its specific method, speed().

-- continue keyword
The continue statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.

This example skips the value of 4:
```
for (int i = 0; i < 10; i++) {
  if (i == 4) {
    continue;
  }
  System.out.println(i);
}
```

— Scanner
To get the user input
java.util package

Methods in Scanner:
nextLine();
nextBoolean();

-- difference between iteration and enumeration in java
1) The main difference between Iterator and Enumeration is removal of the element while traversing the collection.

Iterator can remove the element during traversal of collection as it has remove() method. Enumeration does not have remove() method.
2) Enumeration is fail-safe in nature, means it does not throw ConcurrentModificationException, when the structure is modified during iteration whereas, Iteration is fail fast, it throws the exception when tried to change the collection's structure during iteration.

-- Unicode
Java uses Unicode to represent the characters.

-- Literals
Any constant value that is assigned to a variable is called literal in Java. For example –
// Here 101 is a literal
int num = 101

---CLASS

-- A class is a group of objects which have common properties.
It is a template or blueprint from which objects are created.
It is a logical entity. It can't be physical.

A class in Java can contain:
Fields
Methods
Constructors
Blocks
Nested class and interface

Instance variable in Java
A variable which is created inside the class but outside the method is known as an instance variable.
Instance variable doesn't get memory at compile time. It gets memory at runtime when an object or instance is created.
That is why it is known as an instance variable.

new keyword in Java
The new keyword is used to allocate memory at runtime. All objects get memory in Heap memory
---

-- Abstraction is a process of hiding the implementation details and showing only functionality to the user.

Ways to achieve Abstraction

There are two ways to achieve abstraction in java

1.Abstract class (0 to 100%)
2. Interface (100%)

An abstract class must be declared with an abstract keyword.
It can have abstract and non-abstract methods.
It cannot be instantiated.
It can have constructors and static methods also and even main() method.
It can have final methods which will force the subclass not to change the body of the method.

The final keyword in java is used to restrict the user. for security purpose.
final can be a variable, method, class, parameter.

final variabe:

---

1. If you make any variable as final, you cannot change the value of final variable(It will be constant).
2. A final variable that is not initialized at the time of declaration is known as blank final variable or uninitialized final variable.
3. We can initialize blank final variable only in constructor.
4. A static final variable that is not initialized at the time of declaration is known as static blank final variable.
It can be initialized only in static block.

final method:

---

1. final method cannot be overriden, but can be inherited.

final class:

A class declared as final cannot be iherited by any subclass

final parameter:

---

cannot change the value of the parameter

Constructors cannot be declared as final , because by default, constructors are not inherited (only invoked using super keyword in subclass).
So no need to declare them as final

Example : Wrapper Classes like String,Integer,Float are declared final , no one can override these.

INNER CLASS

Java Inner class or Nested class is a class which is declared inside the class or interface.

Advantage of java inner classes:

---

1) Nested classes represent a special type of relationship that is it can access all the members (data members and methods) of outer class including private.
2) Nested classes are used to develop more readable and maintainable code because it logically group classes and interfaces in one place only.
3) Code Optimization: It requires less code to write.

Difference between nested class and inner class in Java:

Inner class is a part of nested class. Non-static nested classes are known as inner classes.

Types of Nested classes

---

1. Non-static nested class (inner class)
    1. Member inner class
    2. Anonymous inner class
    3. Local inner class
2. Static nested class

MEMBER INNER CLASS

A non-static class that is created inside a class but outside a method is called member inner class.

A static class created inside a class is called static nested class.
It can access only static members of the outer class inclding private.

Cannot access the non static members of the outer class

A class i.e. created inside a method is called local inner class in java.
If you want to invoke the methods of local inner class, you must instantiate this class inside the method.
Rules:
1. Local variable can't be private, public or protected.
2. Local inner class cannot be invoked from outside the method.
3. Local inner class cannot access non-final local variable till JDK 1.7.
   Since JDK 1.8, it is possible to access the non-final local variable in local inner class.

A class that have no name is known as anonymous inner class in java.
It should be used if you have to override method of class or interface.
Advantage: Instead of creating a separate child class to override the super class methods, we use anonymous inner class with no class name to override

Java Anonymous inner class can be created by two ways:
1. Class (may be abstract or concrete).
2. Interface

OBJECT:

--- An Object is an entity which has state and behaviour e.g., chair, bike, marker, pen, table, car, etc.
It can be physical or logical (tangible and intangible).
The example of an intangible object is the banking system.

An object has three characteristics:

State: represents the data (value) of an object.
Behavior: represents the behavior (functionality) of an object such as deposit, withdraw, etc.
Identity: it is used internally by the JVM to identify each object uniquely.

An object is an instance of a class. A class is a template or blueprint from which objects are created. So, an object is the instance(result) of a class.

--- Object Definitions:

An object is a real-world entity.
An object is a runtime entity.
The object is an entity which has state and behavior.
The object is an instance of a class.
object gets the memory in heap memory area.

There are 3 ways to initialize object in Java. Initializing an object means storing data into the object.
By reference variable
By method
By constructors

Example:
Student student = new Student("RAM", 101,"CSE"); //Creating object & Object initialization through constructor
Branch branch = new Branch();
branch.branchName = "CSE"; // // Object Initialization through reference variable
branch.branchID = 1;
student.setStudentLastName("XYZ"); // Object initialization through method
new Branch().printGreetings(); //calling method with anonymous object
Labs lab1 = new Labs(), lab2 = new Labs(); // creating multiple objects

There are many ways to create an object in java. They are:

By new keyword
By newInstance() method
By clone() method
By deserialization
By factory method etc.

--- Anonymous object
Anonymous simply means nameless. An object which has no reference is known as an anonymous object. It can be used at the time of object creation only.
If you have to use an object only once, an anonymous object is a good approach.
new Branch().printGreetings(); //calling method with anonymous object

— The objects can be created without new keyword by following ways
Using Class.forName() and Class.newInstance()
ClassLoader loadClass()
Using Object.clone()

Deserialization
Using reflection

-- Create object using Class.newInstance()
Class ref = Class.forName("DemoClass");
DemoClass obj = (DemoClass) ref.newInstance();
Class.forName() loads the class in memory. To create an instance of this class, we need to use newInstance().

-- Create object using class loader's loadClass()
Just like above method, class loader's loadClass() method does the same thing. It creates a new instance of class using an existing instance of same class.

instance.getClass().getClassLoader().loadClass("NewClass").newInstance();

--Create object using Object.clone()
This is also a way to have a new independent instance of a class.
NewClass obj = new NewClass();
NewClass obj2 = (NewClass) obj.clone();

-- Create new object using serialization and deserialization
If you have gone through this article, you can understand that serialization and de-serialization is also a way to have another instance of a class in system.

ObjectInputStream objStream = new ObjectInputStream(inputStream);
 NewClass obj = (NewClass) inStream.readObject();

-- Create new object using reflection [refer reflectionAPI package)
Reflection is also a popular way to create new instances in most of available frameworks.
constructor.newInstance(); or
class.newInstance();

OBJECT CLASS:
The Object class is the parent class of all the classes in java by default.
The Object class is beneficial if you want to refer any object whose type you don't know.

Notice that parent class reference variable can refer the child class object, know as upcasting.

The Object class provides some common behaviors to all the objects such as object can be compared, object can be cloned, object can be notified etc.

Let's take an example, there is getObject() method that returns an object but it can be of any type like Employee,Student etc, w
e can use Object class reference to refer that object. For example:

Object obj=getObject();//we don't know what object will be returned from this method

-- Methods of Object class:
public final Class getClass()
public int hashCode()
public boolean equals(Object obj)
protected Object clone() throws CloneNotSupportedException
public String toString()
notify()
notifyAll()
wait()

Object Cloning
The object cloning is a way to create exact copy of an object. The clone() method of Object class is used to clone an object.
The java.lang.Cloneable interface must be implemented by the class whose object clone we want to create.
If we don't implement Cloneable interface, clone() method generates CloneNotSupportedException.

The clone() method is defined in the Object class.

-- Class.forName()
It loads the class dynamically where we doesn't know the class name before hand. Once the class is loaded we will use newInstance() method to create the object dynamically.
It also executes static block after loading the class
In JDBC, class.forName() method is used to register the Driver class. It loads the driver class dynamically

-- Why to override equals() and hashcode()? Not to allow duplicate objects inserted into the hash based collections
https://www.techiedelight.com/why-override-equals-and-hashcode-methods-java/#:~:text=Overriding%20only%20equals()%20method,on%20each%20of%20the%20two

The hash based collections like HashSet, HashMap, HashTable are organized like a sequence of buckets,
and the hash code value of an object is used
 1. to determine the bucket to store the object.
 2. to search the object position during retrieval , along with hashcode, equals() also used to search

Note:
1. equal objects must have equal hash codes but not vice versa.
2. if two objects are equal according to the equals(Object) method,
then calling the hashCode method on each of the two objects must produce the same integer result.
3. hascode() returns distinct integers for distinct objects

equals() Contract :
The equals() method must be:
reflexive: an object must equal itself
symmetric: x.equals(y) must return the same result as y.equals(x)
transitive: if x.equals(y) and y.equals(z), then also x.equals(z)
consistent: the value of equals() should change only if a property that is contained in equals() changes (no randomness allowed)

-- Wrapper class
A wrapper class converts the primitive data type such as int, byte, char, boolean, etc. to the objects of their respective classes
such as Integer, Byte, Character, Boolean, etc. respectively.

 Use of Wrapper classes:
 Change the value in Method:
 Serialization

Synchronization
java.util package:
Collection Framework

Primitive Type	Wrapper class
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

-- Autoboxing/ boxing
Converting the primitive datatype (int, float, double, char) to the wrapper classes (Integer, Float, Double, Character) object automatically.
Example of Autoboxing:

```
int a=20;
Integer i=Integer.valueOf(a);//converting int into Integer explicitly
Integer j=a;//autoboxing, now compiler will write Integer.valueOf(a) internally, automatically
```

Unboxing: Convert Wrapperclass object to primitive type
Example of Unboxing:

```
Integer a=new Integer(3);
int i=a.intValue();//converting Integer to int explicitly
int j=a;//unboxing, now compiler will write a.intValue() internally, automatically
```

Note: Autoboxing is used in the collections
Ex: ArrayList<Integer> list = new ArrayList<>();
list.add(5); // auotboxing is happening here, conversion of 5 (int) into Integer object
list.add(4);

Note: Since J2SE 5.0, autoboxing and unboxing feature convert primitives into objects and objects into primitives automatically.
The automatic conversion of primitive into an object is known as autoboxing and vice-versa unboxing.

———————

——

OOPS concepts:

1. Abstraction
2. Encapsulation
3. Polymorphism
4. Inheritance

— ABSTRACTION - Abstract Class & Interfaces

ABSTRACTION

Abstraction is a process of hiding the implementation details and showing only functionality to the user.

Mostly, we don't know about the implementation class (which is hidden to the end user), and

an object of the implementation class is provided by the factory method.

--Ways to achieve Abstraction

There are two ways to achieve abstraction in java

1.Abstract class (0 to 100%)

2. Interface (100%)

ABSTRACT CLASS:

An abstract class must be declared with an abstract keyword.

It can have abstract and non-abstract methods.

It cannot be instantiated.

It can have constructors and static methods also and even main() method.

It can have final methods which will force the subclass not to change the body of the method.

Rules:

-- If there is an abstract method in a class, that class must be abstract.

-- If you are extending an abstract class that has an abstract method, you must either provide the implementation of the method or make this class abstract.

Similarities:

Abstract class and interface both are used to achieve abstraction where we can declare the abstract methods.

Abstract class and interface both can't be instantiated.

Abstract class is used to specify if there are common functionality in all the sub

classes.

--DIFFERENCES:
ABSTRACT CLASS
INTERFACE
Abstract class can have abstract and non-abstract methods.
Interface can have only abstract methods. Since Java 8, it can have default and static methods also.
2) Abstract class doesn't support multiple inheritance. Interface supports multiple inheritance.
3) Abstract class can have final, non-final, static and non-static variables.
Interface has only static and final variables.
4) Abstract class can provide the implementation of interface.
Interface can't provide the implementation of abstract class.
5) The abstract keyword is used to declare abstract class. The interface keyword is used to declare interface.
6) An abstract class can extend another Java class and implement An interface can extend another Java interface only
multiple Java interfaces..
7) An abstract class can be extended using keyword "extends". An interface can be implemented using keyword "implements".
8) A Java abstract class can have class members like private, protected, etc.
Members of a Java interface are public by default.

--Difference between Abstract Class and Java 8 Interface

— After having default and static methods inside the interface, we think about the need of abstract class in Java.

An interface and an abstract class is almost similar except that you can create constructor in the abstract class
whereas you can't do this in interface.

— Using Interfaces, can implement multiple inheritance whereas Abstract class cannot be used.

Highest level of Abstraction using Interface.

Interfaces are like standards where all its implementation classes are below the hierarchy.

Note:
Simply, abstract class achieves partial abstraction (0 to 100%) whereas interface achieves fully abstraction (100%).

— When to use an abstract class (ex shape program with area method as same functionality spread across all sub classes)
if we want to declare non-public members. In an interface, all methods must be public.
If we want to add new methods in the future, then an abstract class is a better choice. Because if we add new methods to an interface, then all of the classes that already implemented that interface will have to be changed to implement the new methods.
If we want to provide common, implemented functionality among all implementations of our component, use an abstract class. Abstract classes allow us to partially implement our class, whereas interfaces contain no implementation for any members.

——

INTERFACE
An interface in Java is a blueprint of a class.
Interface fields(variables) are public, static and final by default, and the methods are public and abstract.

Since Java 8, we can have default and static methods in an interface.
Since Java 9, we can have private methods in an interface.

A class that implements an interface must implement all the methods declared in the interface.
Java Interface also represents the IS-A relationship.
It cannot be instantiated just like the abstract class.
--Use of Interface:
1. to achieve abstracton
2. to implement Multiple inheritance
3. to achieve loose coupling

-- Relationship between classes and interfaces
A class extends another class,
A class implements an interface
An Interface extends another interface

-- Multiple inheritance in Java by interface is achieved as follows
A class implements 2 interfaces
an interface extends 2 interfaces

the ambiguity issue is resolved if we use interfaces to implement Multiple inheritance. Since the implementation is provided by the implementation class.

Note: since the methods in interface are public by default,
the overriding methods must be public in order to avoid the access privilege issue.

-- QUESTIONS:
What is marker or tagged interface?
An interface which has no member is known as a marker or tagged interface, for example, Serializable, Cloneable, Remote, etc.
They are used to provide some essential information to the JVM so that JVM may perform some useful operation.

— Nested Interface
An interface i.e. declared within another interface or class is known as nested interface.
Points to remember:
1. Nested interface must be public if it is declared inside the interface but it can have any access modifier if declared within the class.
2. Nested interfaces are declared static implicitly.

-- Question
Can we define a class inside the interface?
Yes, If we define a class inside the interface, java compiler creates a static nested class.

Note:
The abstract class forces the implementation classes to implement the abstract methods present in it.
Hence if any specific mandatory behaviour exist, abstract class can be used to force the user to implement it
This can be achieved using Interfaces too. Interface is more generic and precise.

This class is used when common features are shared by all object.

Why to use Abstract class?
If there are any behaviour or features common across the classes, instead of defining them multiple times,
we have abstract class which declares that feature as abstract method and forces all the implementation classes to implement that

feature .

When to use Abstract class and When to use Interfaces?
If the requirement has class with members having different access modifiers, no need multiple inheritance , need of abstraction ,
sharing common features, need of constructor, then we can go for abstract class
If the requirement has, class with no members, very high level generic functionality, no constructor,
need of multiple inheritance, then we can go for Inheritance

——

— Encapsulation: securing, binding, hiding, protecting
It is a process of wrapping code and data together into a single unit.
We can create a fully encapsulated class in Java by making all the data members of the class private.
Now we can use setter and getter methods to set and get the data in it.
The Java Bean class is the example of a fully encapsulated class.

Advantage:
1. By providing only a setter or getter method, you can make the class read-only or write-only.
2. It provides you the control over the data
3. It is a way to achieve data hiding in Java because other class will not be able to access the data through the private data members.
4.The encapsulate class is easy to test. So, it is better for unit testing.

——

--POLYMORPHISM - Many forms
To perform a single action in different ways.
There are two types of polymorphism in Java: compile-time polymorphism and runtime polymorphism.
We can perform polymorphism in java by method overloading and method overriding.

-- compile time polymorphism.: (method overloading)
If you overload a static method in Java, it is the example of compile time polymorphism.
method overloading is an example of compile time polymorphism.
The method call is decided at compile time, by seeing its arguments

--Runtime Polymorphism (method overriding)
Runtime polymorphism or Dynamic Method Dispatch is a process in which a call to an overridden method is resolved or decided at runtime rather than at compile-time.
In this process, an overridden method is called through the reference variable of a superclass.
The determination of the method to be called during runtime is based on the object being referred to by the reference variable.

--Upcasting
If the reference variable of Parent class refers to the object of Child class, it is known as upcasting.
For upcasting, we can use the reference variable of class type or an interface type
class A{}
class B extends A{}
A a=new B();//upcasting

— Method Overloading
If a class has multiple methods having same name but different in parameters, it is known as Method Overloading.

--Advantage of method overloading
Method overloading increases the readability of the program.

--Different ways to overload the method
There are two ways to overload the method in java
1. By changing number of arguments
2. By changing the data type

--RULES
1. Must have same method name
2. Can have different argument lists or parameters with different data types
3. Can have different return types.Method Overloading is not possible just by changing the return type of the method alone
4. Have different access modifiers.You can vary the access levels in any direction.
5. Throw different checked or unchecked exceptions.
6. You can prevent a method from being overwritten by making it final, or static , but you cannot prevent a method from being overloaded.

diff no of arguments
diff order of dataypes

not with diff return type , method overloading does not happen

— Method Overriding
--If subclass (child class) has the same method as declared in the parent class, it is known as method overriding in Java.
-- Usage
1.Method overriding is used to provide the specific implementation of a method which is already provided by its superclass.
2. Method overriding is used for runtime polymorphism

--Rules for Java Method Overriding
The method must have the same name as in the parent class
The method must have the same parameters or arguments as in the parent class.
The return types must be compatible: Whatever the super class declares as a return type, the subclass should declare either the same type or the sub class type
There must be an IS-A relationship (inheritance).
The method cannot be less accessible. Example: if the super class method is public, you should not make it private while overriding it in subclass

QUESTIONS:
Can we override static method?
No, a static method cannot be overridden. It can be proved by runtime polymorphism, so we will learn it later.

Why can we not override static method?
class memory - static methods,
heap memory - instance methods
It is because the static method is bound with class whereas instance method is bound with an object.
Static belongs to the class area, and an instance belongs to the heap area.

Can we override java main method?
No, because the main is a static method.

-- -- Exceptional Handling with Method Overriding:
Rules :
1. If the superclass method does not declare an exception, subclass overridden method cannot declare the checked exception but it can declare unchecked exception.
2. If the superclass method declares an exception, subclass overridden method can declare same, subclass exception or no exception but cannot declare parent

exception.

-- Difference between Method Overloading and Method Overriding

Method Overloading	Method Overriding
1) Method overloading is used to increase the readability of the program.	Method overriding is used to provide the specific implementation of the method that is already provided by its super class.
2) Method overloading is performed within class.	Method overriding occurs in two classes that have IS-A (inheritance) relationship.
3) In case of method overloading, parameter must be different.	In case of method overriding, parameter must be same.
4) Method overloading is the example of compile time polymorphism.	Method overriding is the example of run time polymorphism.
5) In java, method overloading can't be performed by just changing return type of the method only. Return type can be same or different in method overloading. But you must have to change the parameter.	Return type must be same or covariant in method overriding.

— BINDING:
Binding is a mechanism of creating link between method call and method actual implementation(method body).
As per the polymorphism concept in Java , object can have many different forms. Object forms can be resolved at compile time and run time.

--There are two types of binding
1. Static Binding or Early Binding or Compile time binding : The type of object is determined at Compile time. (Ex: overloading)
If there is any private, final or static method in a class, there is static binding.
2. Dynamic Binding or Late Binding or Run time Binding : The type of the object is determined at Run time (overriding)

Key	Static Binding	Dynamic Binding
Basic	It is resolved at compile time	It is resolved at run time
Resolve mechanism	static binding use type of the class and fields	Dynamic binding uses object to resolve binding
Example	Overloading is an example of static binding	

Method overriding is the example of Dynamic binding
Type of Methods private, final and static methods and variables uses static binding Virtual methods use dynamic binding

——

— Inheritance:
Inheritance in Java is a mechanism in which one object acquires all the properties and behaviors of a parent object.
Inheritance represents the IS-A relationship which is also known as a parent-child relationship.

-- USES:
1. For Method Overriding (so runtime polymorphism can be achieved).
2. For Code Reusability.

-- Terms used in Inheritance
Super Class/Base Class/Parent class
Sub class/child class/Derived class/Extended class

-- The extends keyword indicates that you are making a new class that derives from an existing class. The meaning of "extends" is to increase the functionality.

-- TYPES
There can be three types of inheritance in java:
Single: parent, child
Multilevel: GrandParents, Parents, child
Hierarchical: Parent, child1, child2
In java programming, multiple and hybrid inheritance is supported through interface only not with class

--QUESTION
Why multiple inheritance is not supported in java?
To reduce the complexity and simplify the language, multiple inheritance is not supported in java.
Consider a scenario where A, B, and C are three classes. The C class inherits A and B classes.
If A and B classes have the same method and you call it from child class object, there will be ambiguity to call the method of A or B class.
It can be achieved using interfaces

*/

— Aggregation
If a class have an entity reference, it is known as Aggregation. Aggregation represents HAS-A relationship.
Example
Consider a situation, Employee object contains many informations such as id, name, emailId etc.
It contains one more object named address, which contains its own informations such as city, state, country, zipcode etc.

-- Usage of Aggregation
For Code Reusability.

--When use Aggregation?
Code reuse is also best achieved by aggregation when there is no is-a relationship.
Inheritance should be used only if the relationship is-a is maintained throughout the lifetime of the objects involved;
otherwise, aggregation is the best choice.

——

— Arrays
Java array is an object which contains elements of a similar data type which has contiguous memory location.
Java array is an index based with fixed size.
It inherits the Object class, and implements the Serializable as well as Cloneable interfaces.

Syntax:
int[] numbers;
int numbers[];
int []numbers;

```
int a[] = new int[3]; // declaration and instantiation

a[0] = 10;  // initialization
a[1] = 20;
a[2] = 30;

int a[] = {1,2,3,4,5,6,7} ; // declaration , instantiation and initialization
```

--There are two types of array.
Single Dimensional Array
Multidimensional Array

--Advantages
1. Code Optimization: It makes the code optimized, we can retrieve or sort the data efficiently.
2. Random access: We can get any data located at an index position.
3.type safe : can save specific type of values

--Disadvantages
Size Limit: We can store only the fixed size of elements in the array.
It doesn't grow its size at runtime. To solve this problem, collection framework is used in Java which grows automatically.

-- Multidimensional Array in Java
Data is stored in row and column based index (also known as matrix form).

Example to instantiate Multidimensional Array in Java
int[][] arr=new int[3][3]; //3 row and 3 column

Example to initialize :
arr[0][0]=1;
arr[0][1]=2;
arr[0][2]=3;
arr[1][0]=4;
arr[1][1]=5;
arr[1][2]=6;
arr[2][0]=7;
arr[2][1]=8;
arr[2][2]=9;

-- Jagged Array:
If we are creating odd number of columns in a 2D array, it is known as a jagged array.
In other words, it is an array of arrays with different number of columns.
int a[][] = new int[3][]; // here we are not giving the column size

-- What is the class name of Java array?
In Java, an array is an object. For array object, a proxy class is created whose

name can be obtained by getClass().getName() method on the object.

-- Cloning an Array in Java
Since, Java array implements the Cloneable interface,
we can create the clone of the Java array.
If we create the clone of a single-dimensional array, it creates the deep copy of the Java array.
It means, it will copy the actual value. But, if we create the clone of a multidimensional array, it creates the shallow
copy of the Java array which means it copies the references.

-- Copying a Java Array
- Using arraycopy() method of System class.
System.arraycopy(sourceArray, 0, destinationArray, 0, destinationArray.length);
- Using clone
int destinationArray[] = sourceArray.clone();
- manually
for(int I=0; I<sourcearray.length;i++) {
destinationArray[i] = sourceArray[I]; }

-- Anonymous Array
Array reference that is not stored in a variable.

Ex: new int[]{1,2,3,4};
Ex: sample.min(new int[]{10,22,30,8}); ;//passing anonymous array to method

-- The binarySearch() method of the Arrays class is used to search a specific element in an array. The method uses the binary search algorithm.
The array must be in natural ordering before making this call. It is the simplest and most efficient method to find an element in a sorted array.

——

-- Java String
> String is basically an object that represent sequence of char value
> An array of characters works same as java string

char[] ch = { 'h', 'e', 'l', 'l', 'o' }
String str = new String(ch);
or

String str = "Hello";
In java.lang contain a class String which implements serializable, comparable, charSequence interface

-- CharSequence interface
The CharSequence interface is a readable sequence of char values.
String
StringBuffer
StringBuilder

-- Strings are immutable
The java string is immutable which means it cannot be changed, whenever we change any string a new instance is created
For mutable Strings we can use StringBuffer and StringBuilder class

THere are two ways to create a string object
1. by using String literal
2. by new keyword

1. string literal
using double quotes we can create string
String str = "Hello";
Each time you create a string literal, the JVM check the string constant pool. if the string already exist in the pool
a reference to the pool instance is returned. if the string does not exist in the pool, new string instance is created and placed
in the pool
Note: why java uses the concept of String literal|?
- to make java memory efficient (because no new object is created if it exist already in the string constant pool)

2. by new keyword
String str = new String("Hello")
JVM will create a new string in normal heap memory and the literal hello will be placed in the string constant pool
the variable str will refer to the object in the heap
# String class provides various methods
charAt()
length()
subString()
contains()
join()

equals()
isEmpty()
concat()

-- Immutable string in Java
In java string object are immutable - means unchangeable or unmodifiable
once string object is created its data or state cannot be changed but we can create a new instance

why string objects are immutable in java
because java uses the concept of string literal
suppose there are 10 reference variable all refer to one object "Hello"
if one reference variable changes the value of the object it will affect to all reference variable
that is why string object are immutable in java

-- Compare string
we can compare string in java on the basis of the content and reference
there are 3 ways we can compare string in java
1. using equal() method
2. by using == operator
3. compareTo() method

1. by using equal() method
the string class equals() method compares the original content of the string
it compare value of string for equality
boolean equal(other object) // compare the string to the other object
boolean equalsIgnoreCase(other object) // compare the string to the other string ignoring case

2. by using == operator
the == operator compares references not value

3. using compareTo() method
string class compareTo() method compares values and return an integer that describe if first string is less then equal to
or greater then second string

s1 and s2
s1 == s2 // 0
s1 > s2 // positive

s1 < s2 // negative

-- String concatenation
in java concatenation means a new string that is the combination of multiple string
there are 2 ways we can concat string in java
1. using + operator
2. using concat() method

1. using + operator
in java string concatenation is implemented through the stringBuilder / stringBuffer class and its append method
string concatenation operator process a new string by appending the second operand into the end of the first operand
the string concatenation operator can concat only string but primitive value also

2. using concat() method
string concat method will concatenate the specific string to the end of current string
public String concat(String other)

-- Substring in java
> A substring is a part of string
> java provides a built in method subString() that extract a subString from the given string using index value passed
as an argument
Example:

```
System.out.println("Original message: "+ message);
        System.out.println("SubString starting from index 7: "+
message.substring(7));
        System.out.println("SubString starting from index  0 to 7: "+
message.substring(0, 7));

        String str = new String("Welcome home");
        System.out.println(str);
        System.out.println(str.substring(8));
```

-- Split() method
split() method of a string class can used to extract a substring from a phrase. it accept argument in the form of
regular expression
Ex: String msg = “Hi,Iam,good”;
String arr[] = msg.split(“,”);

arr[0]=Hi
arr[1]=Iam
arr[2]=good
System.*out*.println(Arrays.*toString*(arr));

-- Java StringBuffer class
In java stringBuffer class is used to create mutable (changeable) String object
it is same as string except it is mutable that is it can be changed

StringBuffer class constructor:
StringBuffer() with initial capacity of 16
StringBuffer(String s) - it create an object with specified string
StringBuffer(int capacity) - it create an object with empty string buffer but we can specified capacity as length

# StringBuffer method
append()
insert()
replace()
delete()
reverse()
capacity()
charAt()
length()

insert()
delete()
replace()
reverse()
capacity()

The default capacity of the buffer is 16. if the number of character increase from its current capacity. it increase the
capacity
(old capacity * 2) + 2
default capacity = 16
(16 * 2) + 2 = 34

-- StringBuilder
In java StringBuilder class is used to create a mutable String
the java StringBuilder class is same as StringBuffer class except that it is not

synchronized

StringBuilder has almost all method same as StringBuffer
appemd()
# insert()
# dleete()
# replace()
# reverse()
# capacity()

-- String vs StringBuffer
String
> it is immutable
> String is slow and consume more memory when you concat too many string
> string class override the equal() method of object class , can compare the content of two string by equals() method

StringBuffer
> it is mutable
> it is fast and consume less memory when you concat string
> it does not override the equals() method of the object class

-- StringBuffer and StringBuilder
StringBuffer
> it is synchorinized ie thread safe
> it is less efficient

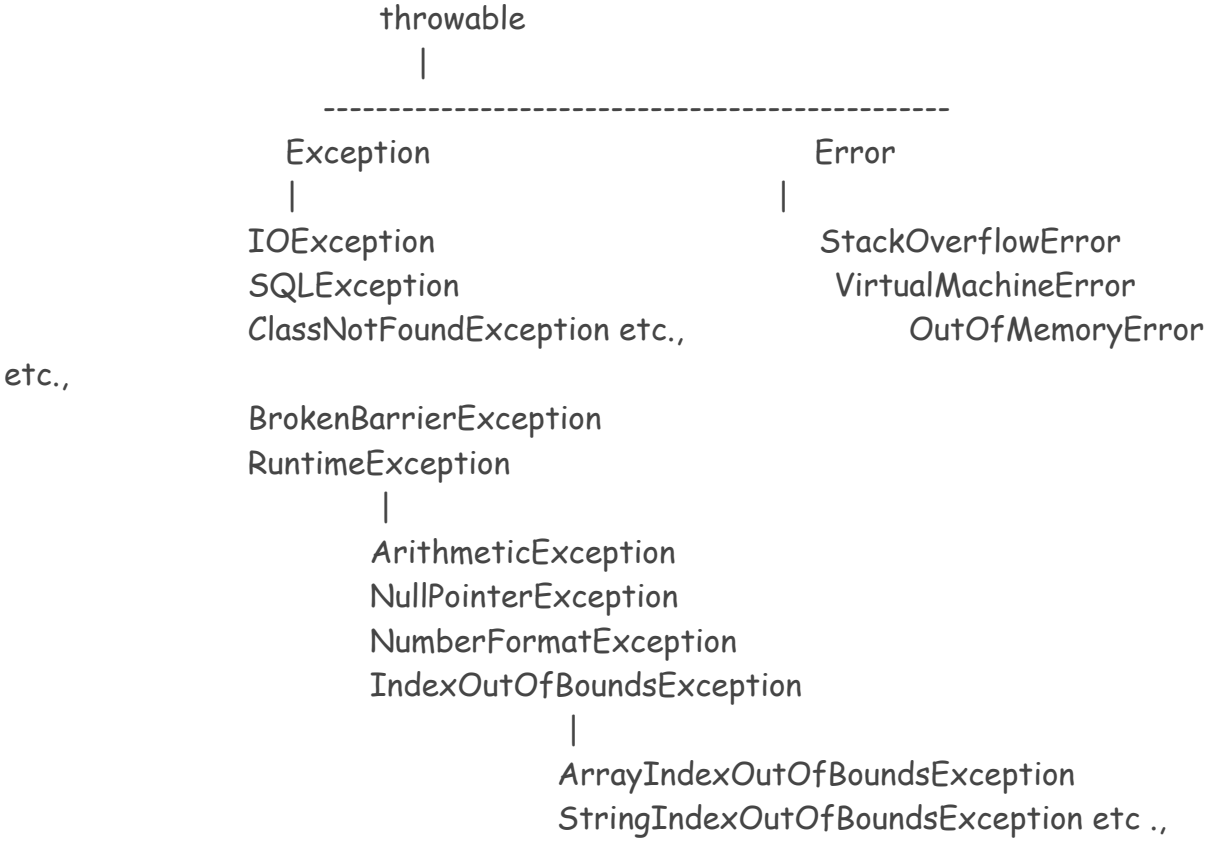
StringBuilder
> it is non-synchorinized , that is it is not thread safe
> it is more efficient

EXCEPTIONAL HANDLING:

-- Exception Handling : Powerful mechanism to handle the runtime errors(ClassNotFoundException, IOException, SQLException, RemoteException, etc.)
to maintain the normal flow of the application.

Advantage : to maintain the normal flow of the application

-- Hierarchy of Java Exception classes



-- Types of Java Exceptions

1.Checked Exception (Compiletime Exception) : The exceptions checked at compile time.

The classes that directly inherit the Throwable class except RuntimeException and Error are known as checked exceptions.

Example: IOException, SQLException, etc.

2. Unchecked Exception (Runtime Exception) : The exceptions not checked at compile time, but they are checked at runtime.

Example: ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException, etc.

3. Error : Error is irrecoverable. Some example of errors are OutOfMemoryError, VirtualMachineError, AssertionError, StackOverflowError etc.

-- Exception Keywords

1. try : The try block contain the exception code or any suspicious code.

The try block must be followed by either catch or finally.

It must be used within the method.

Note: If an exception occurs at the particular statement in the try block, the rest of the block code will not execute.

So, it is recommended not to keep the code in try block that will not throw an exception.

Syntax:

```
try
{
//code that may throw exception
} catch(Exception e) {}

try
{
//code that may throw exception
} finally{}
```

2. catch : To handle the exception by declaring the type of exception within the parameter.

The declared exception must be the parent class exception (i.e., Exception) or the generated exception type.

It must be preceded by try block which means we can't use catch block alone.

It can be followed by finally block later.

You can use multiple catch block with a single try block.

3. finally : To execute the mandatory code.

It is executed whether an exception is handled or not.

4. throw : To throw an exception

5. throws : To declare exceptions.

It specifies that an exception may occur in the method.

It doesn't throw an exception.

It is always used with method signature.

-- Exception Scenarios:

ArithmeticException : int a=50/0

NullPointerException :

```
String a = "null";
System.out.println(s.length());
```

NumberFormatException :

```
String x;
int i= Integer.parseInt(x);
```

ArrayIndexOutOfBoundsException :

```
int arr[] =  new arr[5];
arr[10] = 300;
```

-- Internal Working of Java try-catch block

The JVM firstly checks whether the exception is handled or not.

If handled, rest of the code is executed.

If exception is not handled, JVM provides a default exception handler that performs the following tasks:

Prints out exception description.

Prints the stack trace (Hierarchy of methods where the exception occurred).

Causes the program to terminate.

-- Multi Catch Block

A try block can be followed by one or more catch blocks. Each catch block must contain a different exception handler.

So, if you have to perform different tasks at the occurrence of different exceptions, use java multi-catch block.

Note:

1. At a time only one exception occurs and at a time only one catch block is executed.
2. All catch blocks must be ordered from most specific to most general, i.e. catch for ArithmeticException must come before catch for Exception.

-- Nested Try block : try block inside a try block

Sometimes a situation may arise where a part of a block may cause one error and the entire block itself may cause another error.

In such cases, exception handlers have to be nested.

Note:

1. When any try block does not have a catch block for a particular exception, then the catch block of the outer (parent) try block are checked for that exception, and if it matches, the catch block of outer try block is executed.
If none of the catch block specified in the code is unable to handle the exception, then the Java runtime system will handle the exception. Then it displays the system generated message for that exception.
2. When the try block within nested try block (inner try block 2) do not handle the exception.
The control is then transferred to its parent try block (inner try block 1).
If it does not handle the exception, then the control is transferred to the main try block (outer try block)

where the appropriate catch block handles the exception. It is termed as nesting.

-- finally block: to execute important code such as closing the connection, etc.
The Finally block is always executed whether an exception is handled or not.
Therefore, it contains all the necessary statements that need to be printed regardless of the exception occurs or not.
The finally block follows the try-catch block.
Note: If you don't handle the exception, before terminating the program, JVM executes finally block (if any).

--Why use Java finally block?
To put "cleanup" code such as closing a file, closing connection, etc.
The important statements to be printed can be placed in the finally block.

Note:
1. For each try block there can be zero or more catch blocks, but only one finally block.
2. The finally block will not be executed if the program exits (either by calling System.exit() or by causing a fatal error that causes the process to abort).

--throw keyword: to throw an exception explicitly within a method.
The exception object which is to be thrown must be specified.
The Exception has some message with it that provides the error description.
These exceptions may be related to user inputs, server, etc.
Execution of the current method will stop (the statements after throw won't be executed), and the control will be gone to the caller or the first catch block

Example: throw new exception_class("error message");
        throw new IOException("sorry device error");
            Where the Instance must be of type Throwable or subclass of Throwable.
Note:
If we throw a checked exception using throw keyword, it is must to handle the exception using catch block or the method must declare it using throws declaration.

```
class throwExample4 {
    public static void main(String args[]) {
        try{
```

```
            System.out.println("Task1");
            throw new Exception("Throwing a message"); // generates an Exception
Object with a message
        }catch(Exception e) {
            System.out.println(e);  // prints the error message generated by the
exception object
        }
        System.out.println("rest of the code...");
    }
}
```

-- throws :
throws keyword is used to declare an exception which may occur. DEclared at method signature.
It provides information to the caller of the method about the exception.
It is better for the programmer to provide the exception handling code so that the normal flow of the program can be maintained.
Note: 1. Only Checked exceptions are declared.
The Unchecked exceptions are under the programmer control, can be corrected.
The error are beyond our control.
2. If we are calling a method that declares an exception, we must either catch or declare the exception.

-- Advantage of Java throws keyword
Now Checked Exception can be propagated (forwarded in call stack).
It provides information to the caller of the method about the exception.

-- Can we rethrow an exception?
Yes, by throwing same exception in catch block.

— Diff between throw and throws

throw
1. java throw keyword is used to throw an exception explicitly
2. checked exception cannot be propagated using throw only
3. it is followed by an instance
4. it is used within the method

throws
1. java throws keyword is used to declare an exception
2. check exception can be propagated with throws
3. it is followed by class
4. it is used with the method signature
——————————————————
final vs. finally vs. finalize
1. final
it is used to apply restrictions on variable, method, and class; final class can't be inherited, final method can't be overridden,
the final variable value can't be changed
final is a keyword

2. finally
it is used to place important code, and it will be executed whether an exception is handled or not
finally is a block

3. finalize
it is used to perform clean up just before an object is garbage collected
finalize is method
——————————

-- Exceptional Handling with Method Overriding:
Rules :
1. If the superclass method does not declare an exception, subclass overridden method cannot declare the checked exception but it can declare unchecked exception.
2. If the superclass method declares an exception, subclass overridden method can declare same, subclass exception or no exception but cannot declare parent exception.

-- Custom or User defined Exceptions
Creating our own Exception is known as custom exception or user-defined exception.
Usage:
To customize the exception according to the user need.
To implement Business logic exceptions, so that it gives good understanding for the application users.

-- Throwable is a root class for all the Exception classes and the Error classes.
--Throwable Methods
getMessage() : prints the detailed message given in the Throwable class or its sub calsses
getSuppressed()
getStackTrace()
fillInStackTrace()
getLocalizedMessage()
initCause()
getCause()
printStackTrace() : prints the class name, method, line number where the exception occurred.
The method printStackTrace prints the stack of methods that have been called when the exception has occured.

-- Exception Propagation
An exception is first thrown from the top of the stack and if it is not caught, it drops down the call stack to the previous method.
If not caught there, the exception again drops down to the previous method, and so on until they are caught or until they reach the very bottom of the call stack. This is called exception propagation.

Note: By default Unchecked Exceptions are forwarded in calling chain (propagated).
By default, Checked Exceptions are not forwarded in calling chain (propagated).

— FIle I/O

— Stream :
A stream can be defined as a sequence of data. There are three kinds of Streams

1. Byte Stream
Operates on Bytes(bits)
FileOutputStream : to create a file and write data into it.
FileInputStream : to read data from the file

2. Character Stream
Operates on characters (16 bit unicode)
FileWriter class : to write characters to a file
FileReader class : to read characters from the file

3. Buffered Stream
Operates line by line
BufferedInputStream or BufferedReader : to read lines from the file
BufferedOutputStream or PrintWriter : to write lines to the file

Note: Internally FileReader uses FileInputStream and FileWriter uses FileOutputStream but here the major difference
is that FileReader reads two bytes at a time and FileWriter writes two bytes at a time.

-- File Operations:
Create File
Get File Information
Write to a file
Read into a file
Delete a file

-- NIO Nonblocking I/O
It takes the data in buffer and reads as junks ... for faster performance

---
