JAVA – NOTES:

WHAT IS JAVA?

* High level
* Object oriented programming language
* Both compiler and interpreted language
* Platform independent
* WRITE ONCE, RUN ANYWHERE

DEVELOPER- **JAMES GOSLING** AT SUN MICRO SYSTEMS IN 1995.

WHY JAVA?

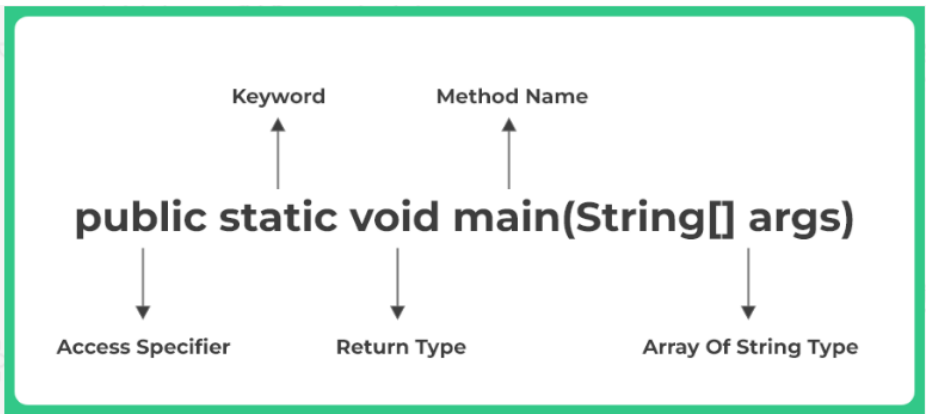
* Platform independent
* Object oriented
* Robust - garbage collection
* Multithreaded
* Simple

DOMAINS - web applications, mobile apps, enterprise systems, backend services, software development.

C++	vs	JAVA
Platform dependent		Platform independent
memory management manual		automatic(garbage collection)
Supports pointer		not have pointers
compiled language		compiled and interpreted

Frist Program:

```
class HelloWorld{
public static void main(String  args[]){
System.out.print("Hello world!!");
}
}
```



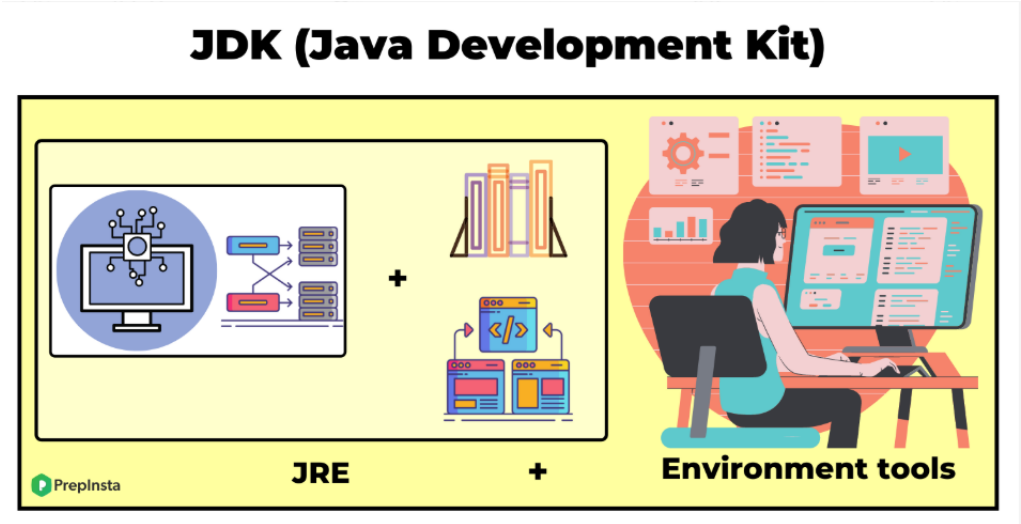
ARCHITECTURE OF JAVA:

1. JDK -JAVA DEVELOPMENT KIT

* compiler + JRE

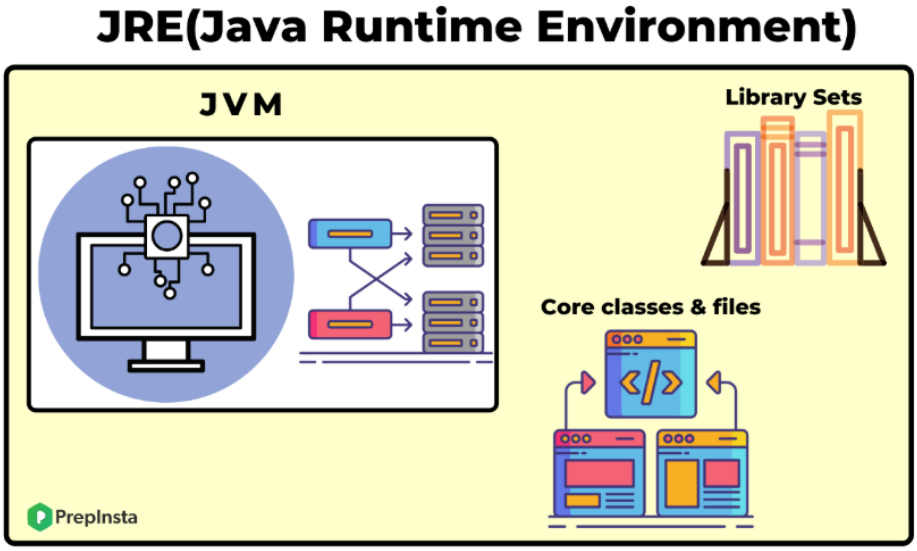
* used for developing java based software

* It contains java compiler, debuggers and deployment tools.



2. JRE- JAVA RUNTIME ENVIRONMENT

* JVM + library sets + core classes and supporting files.

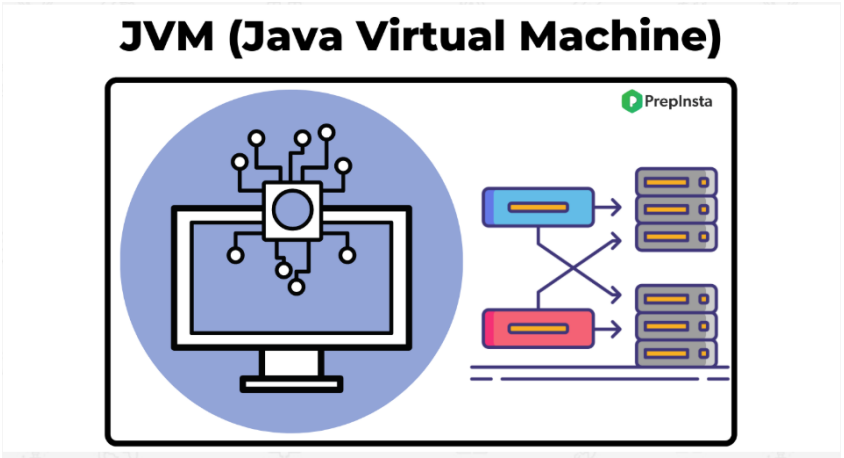


3. JVM - JAVA VIRTUAL MACHINE (platform dependent)

* Core of Java Programming (Heart of Java architecture.)

* When you compile a Java program, the Java compiler converts your code into bytecode, which is then executed by the JVM.

* loading the code, verifying it, implementing it.



- ✓ class loader
- ✓ Byte code verifier
- ✓ JIT - Just In time compiler

HOW A JAVA PROGRAM RUNS?

write code ---> .java file ---> compiler(javac)----> .class file ---> JVM(class loader->bytecode verifier->JIT compiler)---> Code runs

List Of Keywords

abstract	assert	boolean	break
byte	case	catch	char
class	const	continue	default
do	double	else	enum
extends	final	finally	float
for	go to	if	implements
import	instance of	int	interface
long	native	new	package
private	protected	public	return
short	static	strictfp	super
switch	synchronized	this	throw
throws	transient	try	void

DATA TYPES

1. **Primitive datatypes** - int short double char long float byte Boolean.

PRIMITIVE DATA TYPES - default values

Data Type	Default Value	Default Size
Boolean	False	1 bit
Char	'\u0000'.	2 Byte
Byte	0	1 byte
Short	0	2 byte
Int	0	4 byte
Long	0L	8 byte
Float	0.0f	4 byte
Double	0.0d	8 byte

2. **Non primitive datatypes** - strings arrays classes and interfaces

variable:

datatype variable_name = value;

int a = 10,b = 20;

char value = "a";

int myAge = 21;

Local - declared inside a method

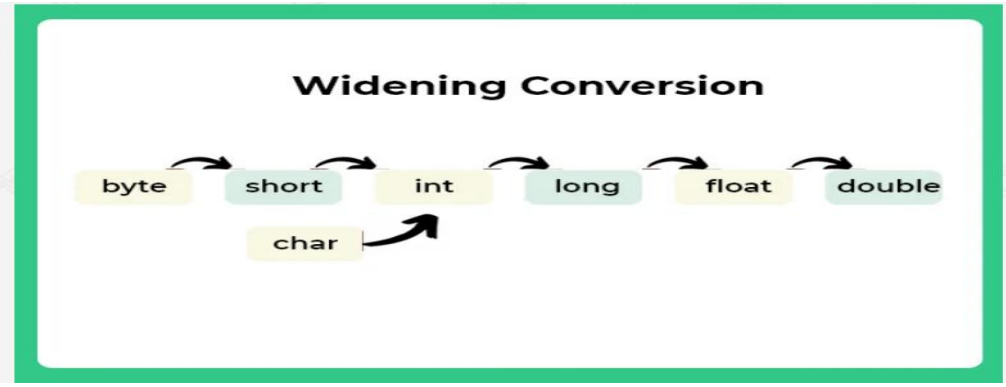
static/ class variable - declared inside outside a method.... using static

Type casting:

1. Implicit casting
2. Explicit casting

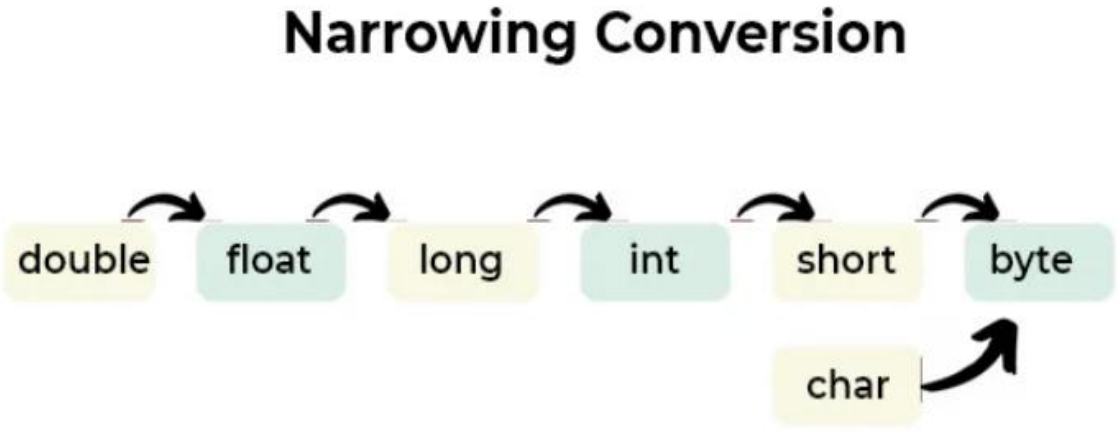
Type conversion:

1. widening(Implicit)- casting automatic



```
public class Main
{
   public static void main(String[] args)
   {
      int num = 500;
      long l = num;
      float f = l;
      System.out.println("Int type value "+num);
      System.out.println("Long type value "+l);
      System.out.println("Float type value "+f);
   }
}
```

2. narrowing(Explicit)- 2 casting manual, need type caster ()



```
class Main
{
   public static void main(String[] args)
   {
      double d = 560.025;
      long l = (long)d;
      int num = (int)l;
      System.out.println("Double type value "+d);
      System.out.println("Long type value "+l);
      System.out.println("Int type value "+num);
   }
}
```

DAY -3

OPERATORS:

- Arithmetic
- Assignment
- Comparison
- Logical
- Bitwise
- Miscellaneous - Ternary operator

A ternary operator can be seen as an alternative to the if-else statement. It works the same as an if-else statement. It evaluates a test condition and executes the block of code accordingly.

If the condition is true, expression1 is executed.

If the condition is false, expression2 is executed.

condition ? expression1 : expression2;

CONTROL STATEMENTS:

1. CONDITIONAL STATEMENTS

if

if (condition) {

// code to be executed if condition is true

}

```
class Main{
   public static void main(String[] args) {
        int income = 5000;
        // checks if income is greater than 1000

        if (income > 1000)
        {
            System.out.println("Hey Prepster. Good going!");
        }

        System.out.println("Welcome to PrepInsta");
  }
}
```

Run

if - else

if (condition) {

// code if condition is true

} else {

// code if condition is false

}

```
public class Main
{
        public static void main(String[] args) {
            int x = 150;
            if (x<100)
                {
                    System.out.println("Hey Prepster, This is the if block");
                }
            else
                {
                    System.out.println("Hey Prepster, This is the else block");
                }
        }
}
```

Run

nested if-else

```
if(condition){
 if(condition{
   //code
 }else{
   //code
 }
}else{
//code
}
```

```java
public class Main
{
        public static void main(String[] args) {
                int a= 100;
                if(a<250)
                {
                    if(a<150)
                    System.out.println("Hey Prepster, Happy Learning");
                    else
                    System.out.println("Keep going, keep learning");
                }

                else
                System.out.println("Welcome to Prepinsta");

        }
}
```

Run

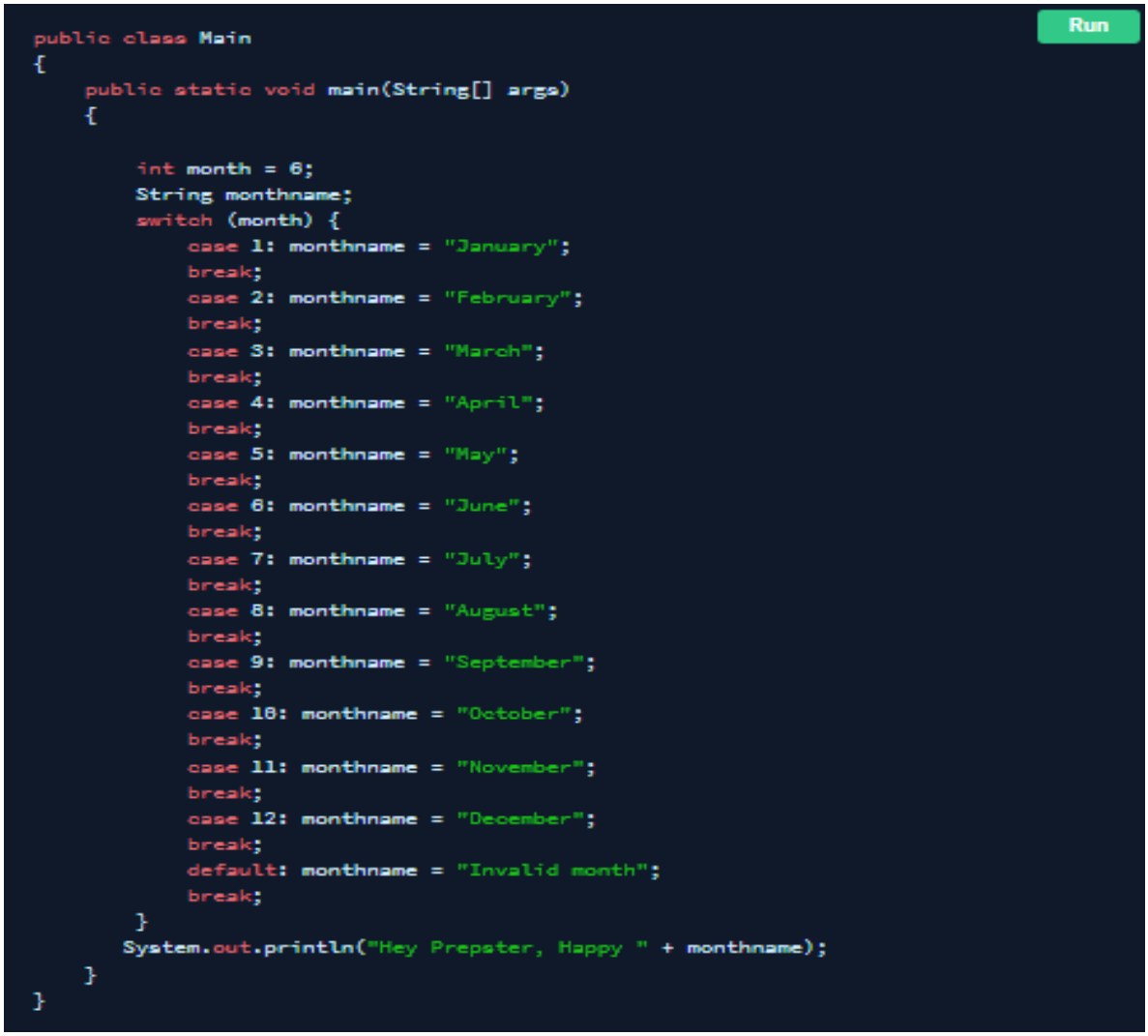
switch statement

```
switch (expression) {
case value1:
// code block
break;
case value2:
// code block
break;
```

default:

// code block

}



2. LOOPING STATEMENTS

for loop

for (initialization; condition; increment/decrement) {

// code to be executed

}

```java
public class Main
{
        public static void main(String[] args) {
             for(int i = 0; i < 5; i++)
                System.out.println("Number:"+i);
        }
}
```

Run

Output:

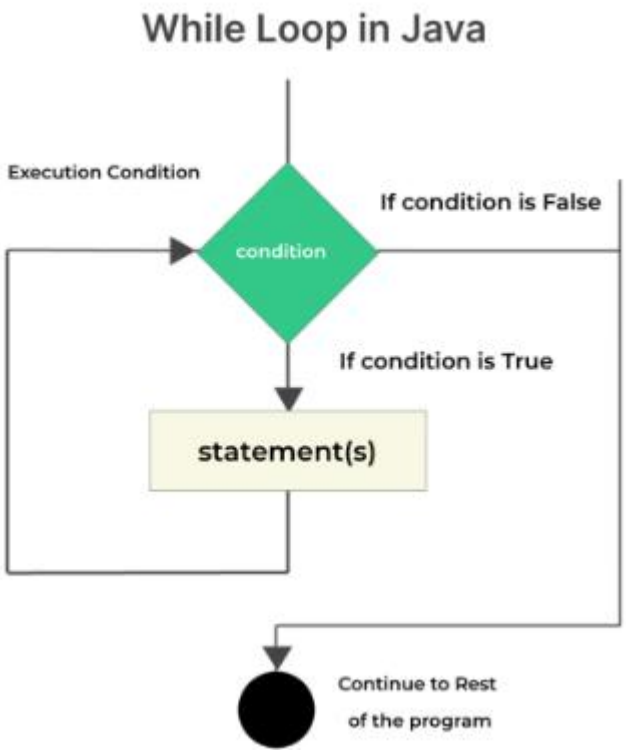
```
Number:0
Number:1
Number:2
Number:3
Number:4
```

while loop

while (condition) {

// code to be executed

}

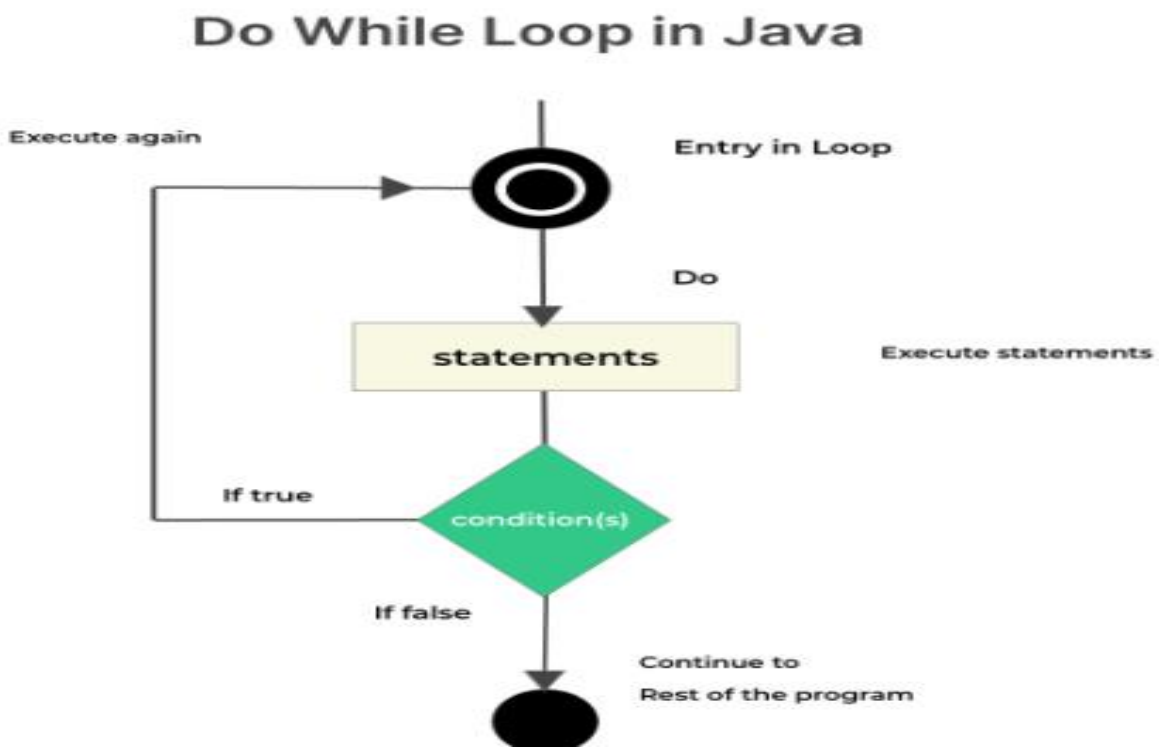


do while loop

do {

// code to be executed

} while (condition);



3. JUMPING STATEMENTS

Break

```
public class Main
{
    public static void main(String[] args)
    {
        for(int i=1;i<=10;i++)
        {
            if(i==5)
            {

             break;  //using break to terminate the loop abruptly
            }
        System.out.println(i*2);
        }
    }
}
```

Run

In this code we initialize a loop which runs until i <= 10. We want the loop to stop executing when i = 5.

Hence we use the *break* statement. Let us see what will be the output of the code above.

Output

```
2
4
6
8
```

Continue

```
public class Main
{
   public static void main(String args[])
   {
        for (int num=0; num<=10; num++)
           {
           if (num==5)
             {
                    continue;
             }

           System.out.print(num+" ");
            }
   }
}
```

Run

n this code we initialize a loop which runs until num <= 10. We want the loop to skip the iteration num = 5.

Hence we use the *continue* statement. Let us see what will be the output of the code above.

The code will yield output as follows, skipping the number ' 5 '

Output

```
1  2  3  4  6  7  8  9  10
```

SCANNER CLASS:

- used to take input from the user or console.
- it is the part of java.util package

import java.util.Scanner;

import java.util.*;

Scanner scan = new Scanner(System.in);

int a = scan.nextInt();

- ✓ nextInt() - integer
- ✓ nextDouble() - double
- ✓ nextLine() - string
- ✓ next() - single word

```java
import java.util.Scanner;   //Importing Class

class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);    //Creating object of scanner class
        System.out.println("Hello! Can you please tell me your name?");
        String obj = sc.nextLine();    //Variable Initialization
        System.out.println("Nice to see you "+obj);
    }
}
```

Run

Input

PrepInsta

Output

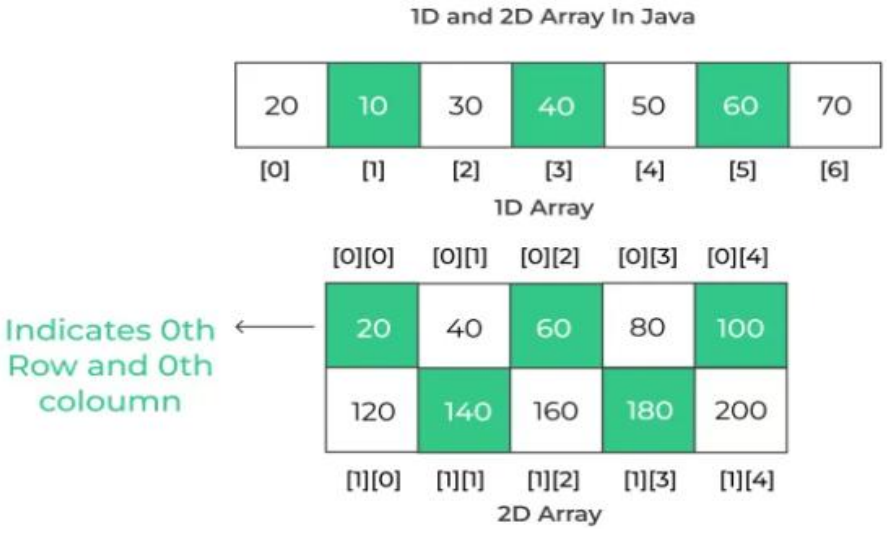
Hello! Can you please tell me your name?

Nice to see you PrepInsta

Arrays:

- 1D - linearly arranged
- 2D - grid like pattern (matrix)

➢ accesed by using index values... 0 to n-1 5--> 0,1,2,3,4



Syntax:

```
datatype arrayname [] = new datatype[array_size];
int a [] = new int [5];
int a[] = {1,2,3,4,5};
int a [] [] = new int [2] [3];
int a [] [] = { {1,2,3},
                {4,5,6}
              };
```

OOPS - classes and objects

Classes:

--> It is a blueprint for creating an object.

--> It contains methods, attributed(variables).

Syntax:

```
<access-modifier> class class\_name{
//attributes & methods
}
```

Objects:

--> Objects are instances of a class.

--> Everything in Java is built upon objects and classes.

Syntax:

```
class_name object_name = new class_name();
```

Methods:

--> block of code that perform specific tasks.

--> They define the behaviour of a clss.

Syntax:

```
returnType method_name(parameters){
//code for execution
}
```

Types:

1. **Instance methods** --> It belongs to a specific object of the class, it can only be called after creating an object.
2. **Static methods** --> It belongs to the class, not to the objects, it can be called without creating an object.

   It is declared using "static" keyword.

   It can be called directly using the class name.

   It can not access non-static variables.
3. **Parameterized methods** --> It is used to provide values inside the methods for operation.

Constructors:

--> It is used to initialize an objects.

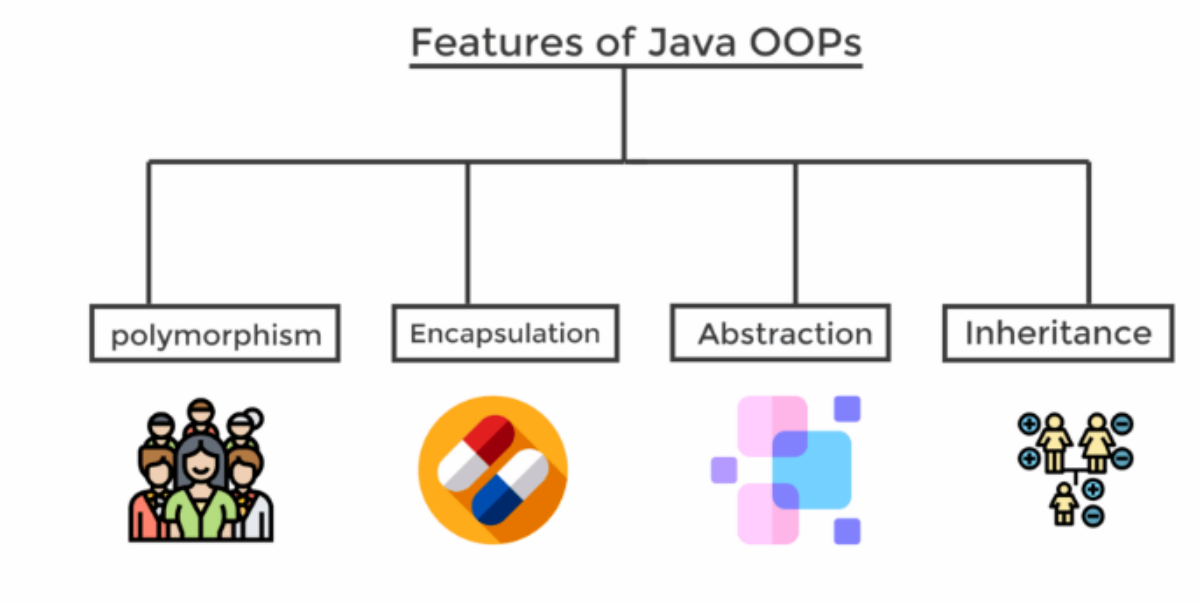
--> It has the same name as the class name.

--> It does not have a return type, if it have a return type, then it is treated like a methods.

Types:

1. Default constructor
2. Parameterized constructor

4-pillars of OOPS:



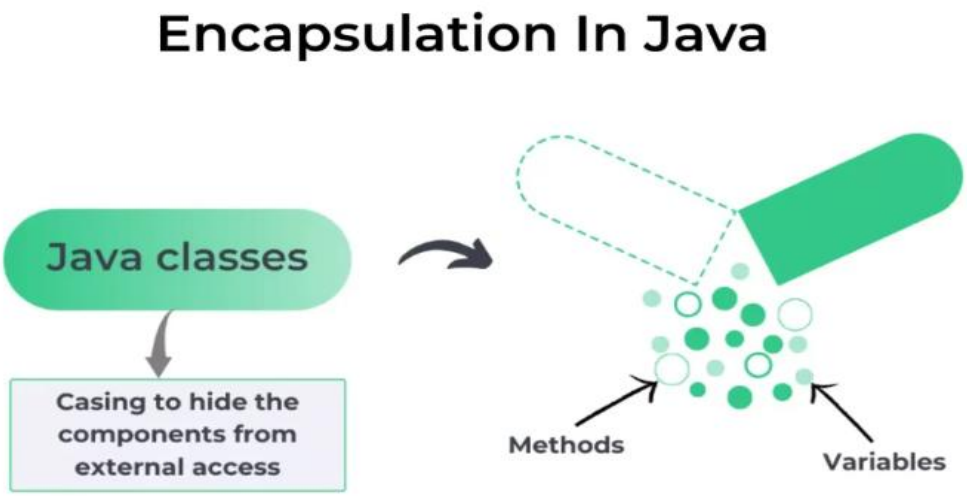
1. Encapsulation:

--> Bundling of data and methods into an single unit, that is class.

--> Process of hiding the internal details of an object from the outside word.

Eg: Bag--> class, items inside the bag --> data, Zipper--> controllers to access the data.

--> Encapsulation is achieved using the access modifiers.



Access modifiers: (4 types)

1. Public ----> allows a class, method or variable to be accessed from any other classes in the same package or the different packages.
2. Private ----> accessible only within the same class.
3. Protected ----> accessible within the same class and its subclasses, even in different packages.
4. Default ----> accessible only within the same package.

Getters and Setters:

--> In Encapsulation, the getter and setter methods are used to access and update private variables of a class.

---> Getter- used to read the value of a private variable, Setter - used to modify or set the value to the private variable.

```java
class Person {
    private String name;
    private int age;

    // Getter methods
    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }

    // Setter methods
    public void setName(String name) {
        this.name = name;
    }

    public void setAge(int age) {
        this.age = age;
    }
}

public class Main {
    public static void main(String[] args) {
        Person person = new Person();

        // Set the name and age using the setter methods
        person.setName("John");
        person.setAge(30);

        // Get the name and age using the getter methods
        String name = person.getName();
        int age = person.getAge();

        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
    }
}
```

Output

```
Name: John
Age: 30
```

this - keyword:

--> It reference to the current object.

--> This is commonly used to resolve naming conflicts.

static - keyword:

--> It is used for memory management.

--> It an be applied to variable, methods and nested classes.

--> Static members belongs to class rather than any specific instance.

final - keyword:

-->It is used to declare constants, prevents method overriding and prevent inheritance.

--> Once a variable is declared as final, its value cannot be changed.

--> final variables : value cannot be changed

final methods : cannot be overridden , final classes : cannot be inherited.

2. Inheritance:

--> It is a mechanism, where one class can inherit fields and methods from another class.

--> This allows for the creation of a new class based on an existing class.

--> Enabling code reuse.

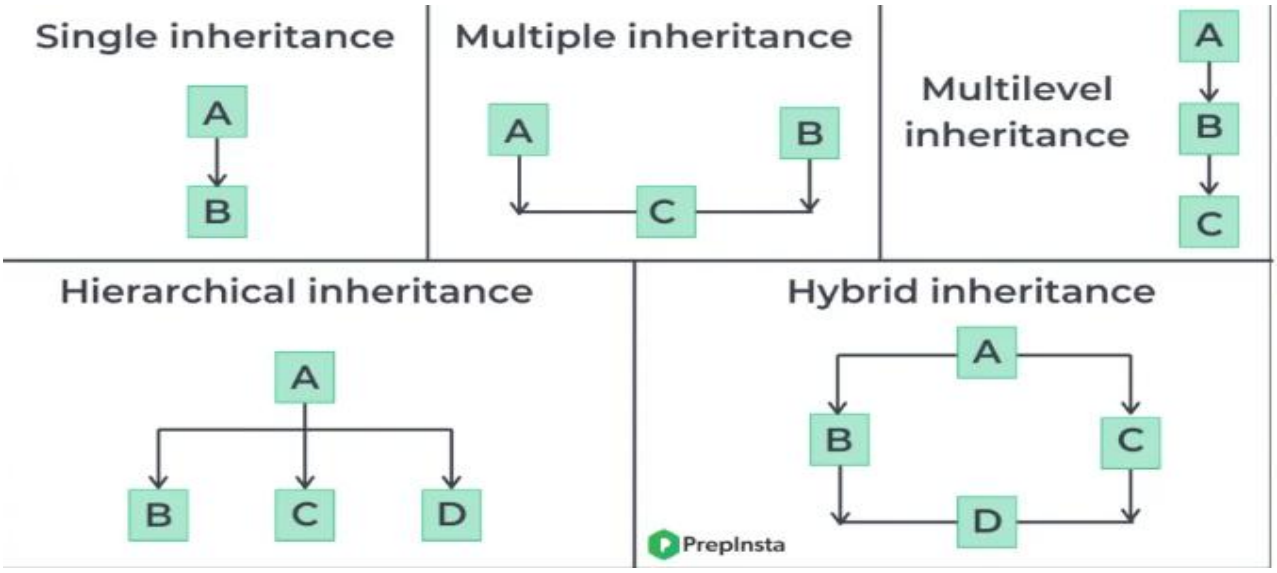
--> Establish a relationship between classes.

--> Inheritance --- extends

Super Class---> Parent Class

Sub Class ---> child Class

Types:



super- keyword:

--> It refers to the immediate parent class.

--> It is used to access parent class methods, variables or constructors from a subclass.

4. Abstraction:

--> Data abstraction is the process of hiding certain details and displaying only essential information to the user.

--> Abstraction is achieved by interfaces and abstract class.

--> Using interfaces we can achieve 100% abstraction in our code.

--> **Abstract class:**

- cannot be instantiated
- It contains both the abstract method and also the non-abstract methods.
- It is declared by using the "abstract" keyword.
- They have constructors, members and variables.

--> **Abstract methods:**

- Can be declared only inside the abstract class.
- It has no body.
- The body is provided by a subclass which inherits the abstract class.

```
import java.util.*;

// class definition
abstract class Human {

    // abstract function
    public abstract void Sound();
    public void sleep() {
      System.out.println("Zzz");
    }
  }

// defining extended class
class Men extends Human {
    public void Sound() {
      System.out.println("The Men says: PrepInsta");
    }
  }

class Main {
    public static void main(String[] args) {
      // Object of men class
      Men myMen = new Men();

      // function calling
      myMen.Sound();
      myMen.sleep();
    }
  }
```

```
Output:

The Men says: PrepInsta
Zzz
```

5. Interface:

--> It contains only the abstract methods.

--> It supports multiple inheritance in java.

--> It also contains static final constants.

--> It is declared using "interface" keyword.

--> All methods in an interface are public and abstract.

--> Interface -- implements.

```
interface PrepInsta
{
    public void method1();
    public void method2();
}

class Main implements PrepInsta
{
    public void method1()
    {
         System.out.println("Here is method1 by Interface");
    }
    public void method2()
    {
         System.out.println("Here is method2 by Interface");
    }
    public static void main(String arg[])
    {
         PrepInsta obj = new Main();
         obj.method1();
         obj.method2();
    }
}
```

6. Polymorphism:

--> Polymorphism (Greek word) Poly ---> many , morph ---> form.

--> It refers to the ability of objects of different classes the be treated as if they are objects of the same class.

Types:

1. **Compile time polymorphism:**

--> Static or method overloading.

--> class has multiple methods with same but they differ with the count of parameters or the type of parameters.

```
public class Main {

        public int sum(int x, int y)
        {
                return (x + y);
        }

        public int sum(int x, int y, int z)
        {
                return (x + y + z);
        }

        public static void main(String args[])
        {
                Main s = new Main();
                System.out.println(s.sum(20,10));
                System.out.println(s.sum(16, 78, 4));
        }
}
```

2. Runtime polymorphism:

--> Dynamic polymorphism, method overriding.

--> The superclass can be overridden in a subclass to provide a specific implementation.

```
class Parent {
    public void walk() {
        System.out.println("I walk slowly");
    }
}

class Child extends Parent {
    @Override
    public void walk() {
    System.out.println("I walk faster");
    }
}


public class Main {
    public static void main(String args[])
    {
        Parent pObj = new Parent();
        pObj.walk();
        Parent cObj = new Child();
        cObj.walk();
    }
}
```

Output:

I walk slowly

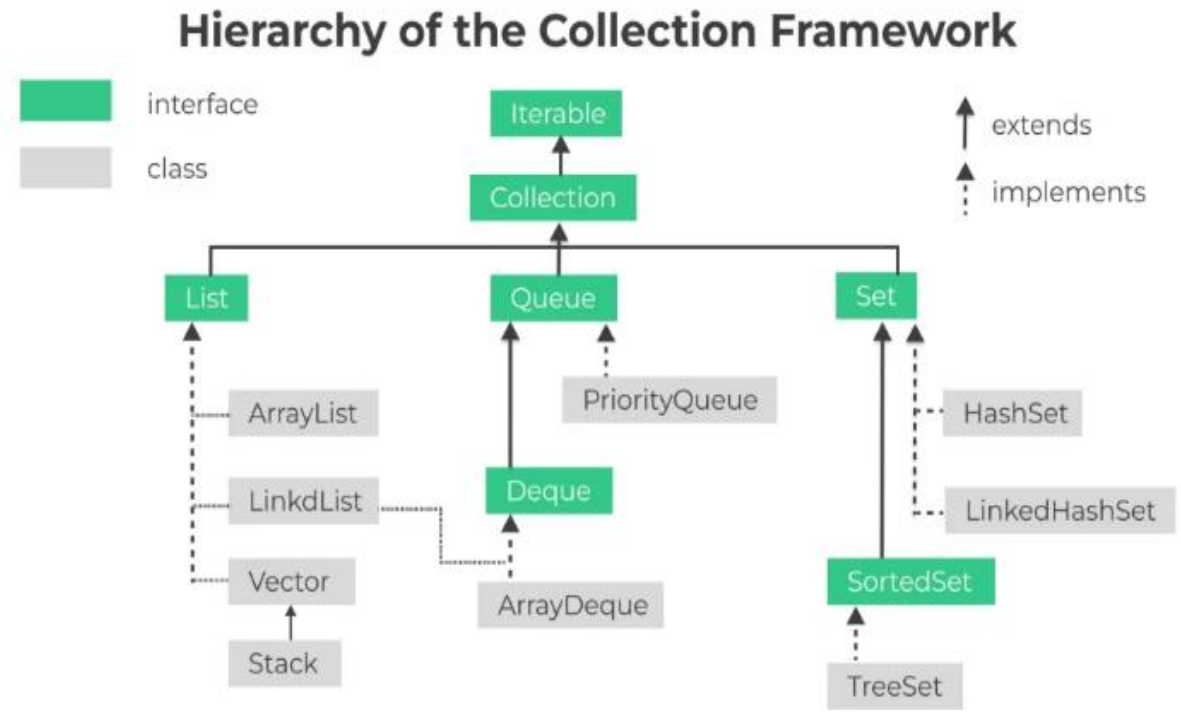
I walk faster

Collections:

Java Collection Framework (JCF) is a set of classes and interfaces that provide ready-made data structures to store and manipulate groups of objects efficiently.

Java provides built-in collection classes like List, Set, Map and Queue, so developers don't need to write their own data management algorithms.

The Collection Framework improves productivity by making code more reusable, maintainable and faster to develop.



Interfaces:

1. **List** --> ArrayList, LinkedList, Stack, Vector
   List interface in Java extends the Collection interface and is part of the **java.util** package. It is used

to store ordered collections where duplicates are allowed and elements can be accessed by their index.

Key Features

- Maintains insertion order
- Allows duplicate elements
- Supports null elements (implementation dependent)
- Provides index-based access

2. **Set** --> HashSet, LinkedHashSet, TreeSet
   In Java, the Set interface is a part of the Java Collection Framework, located in the **java.util** package. It represents a collection of unique elements, meaning it does not allow duplicate values. Most Set implementations allow only a single null element.

3. Queue --> PriorityQueue

The Queue Interface is a part of **java.util** package and extends the Collection interface. It stores and processes the data in an order where elements are added at the rear and removed from the front.

Key Features

- FIFO Order : Elements are processed in the order they were inserted (First-In-First-Out).
- No Random Access : Unlike List, elements cannot be accessed directly by index.

4. Map --> TreeMap, LinkedHashMap, HashMap

In Java, the Map Interface is part of the **java.util** package and represents a **collection of key-value pairs**, where:

- No Duplicate Keys : Keys should be unique, but values can be duplicated.
- Null Handling : HashMap and LinkedHashMap allow one null key, and TreeMap does NOT allow null keys (if natural ordering is used).

Packages:

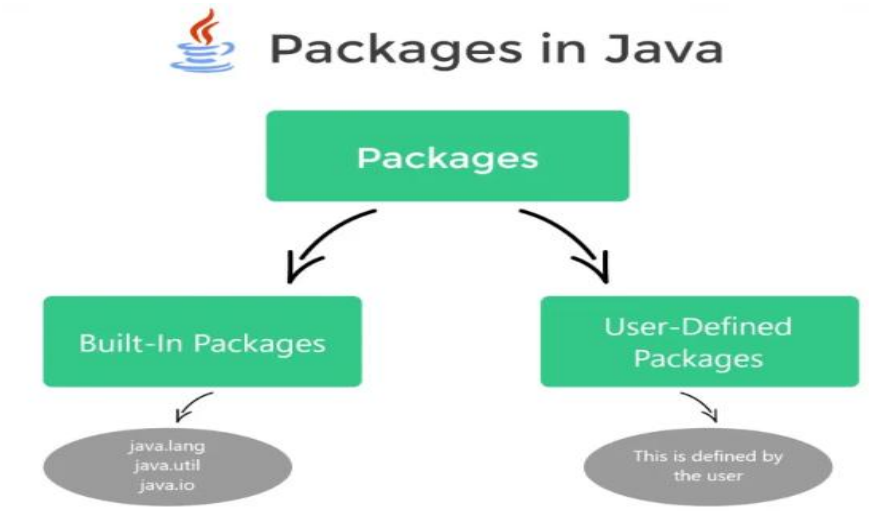
--> It is nothing but a organizer, like a directories in our PC.

--> It contains classes , interfaces and sub-packages.

--> Used to avoid naming conflicts and gives control access with the use of access modifiers over classes.

--> Namespace management, Access Protection, Organizational structure, reusability.

Types:



Memory Management:

1. Memory Areas in Java

Java uses different memory spaces to manage data:

• **Heap Memory:** Stores all **objects** created in a program. Managed automatically by the **JVM.(values)**

• **Stack Memory:** Stores method calls and **local variables.** Each thread gets its own stack.(variable name)

• **Method Area:** Stores class information such as class names, static variables, and constants.

2. Garbage Collection

Java cleans up memory automatically using Garbage Collection.

Key points:

• No need to manually free memory

• Removes objects that are no longer used

• Works in generations: Young objects and old objects are handled differently for efficiency

• Uses algorithms like Mark & Sweep and Copying to reclaim memory safely

Main benefit: Helps prevent memory buildup and crashes

Why Memory Management Matters

• Faster performance and quicker response time

• Uses memory efficiently so applications scale better

• Prevents bugs and crashes caused by memory issues

• Better user experience overall

Lambda expression:

Interface:

--> It contains only the abstract methods.

--> It supports multiple inheritance in java.

--> It also contains static final constants.

--> It is declared using "interface" keyword.

--> All methods in an interface are public and abstract.

--> Interface -- implements.

--> object cannot be created.

Types - 3:

1. **Normal interface -** with multiple abstract methods.
2. **Functional interface -** with only one abstract method --> lambda expression
3. **Marker interface -** does not contains any methods in it.

Lambda expression:(comes with functional interface)

--> short way to represent the anonymous function(a function with out a name).

--> Less code.

Stream API:

--> Introduced in Java 8.

--> Its a sequence of elements.

--> It is a part of java.util.stream.

--> Processed using operations like filter, map, reduce etc, etc.

--> It doesn't store data but performs computations on it.

Task - have to print the names starts with letter "A"

without streams:

```
List <String> names = Arrays.asList("Ajith", "Vijay", "Peter", "Akshaya");
for(String name :names){
 if (name.startsWith("A")){
  System.out.println(name);
  }
}
```

with streams:

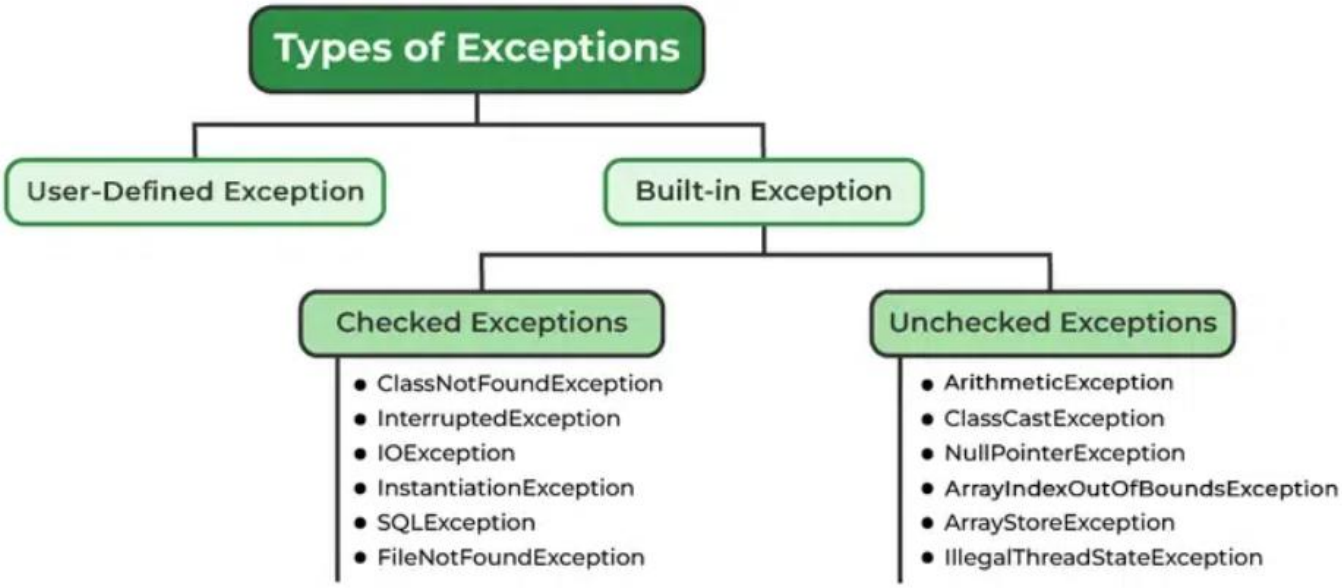
```
List <String> names = Arrays.asList("Ajith", "Vijay", "Peter", "Akshaya");
names.stream().filter(name->
name.stratsWith("A")).forEach(System.out::println);
```

Filter operation: Filters the elements based on certain conditions.

Map operation: Transform the elements.

Reduce operation: Aggregates the elements into a single unit.

Exceptions in Java:



Types of Java Exceptions

There are mainly two types of exceptions: checked and unchecked. An error is considered as the unchecked exception. However, according to Oracle, there are three types of exceptions namely:

1) Checked Exception

The classes that directly inherit the Throwable class except RuntimeException and Error are known as checked exceptions. For example, IOException, SQLException, etc. Checked exceptions are checked at **compile-time.**

2) Unchecked Exception

The classes that inherit the RuntimeException are known as unchecked exceptions. For example, ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException, etc. Unchecked exceptions are not checked at compile-time, but they are checked at **runtime.**

3) Error

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

Important components:

1. try{ }
2. catch{ }
3. finally{ }

```java
import java.util.*;

public class Main{
    public static void main(String args[]){
        // try block
        try{
            // try block Statements
            int temp = 50/5;
            System.out.println(temp);
            System.out.println("I am in the try Block");
        }
        // catch block
        catch(NullPointerException e){
                        //catch block exceptions
            System.out.println(e);
        }
        // finally block
        finally {
            // finally blick statement
            System.out.println("I am in finally block");
        }
    }
}
```

```
Output:

10
I am in the try Block
I am in finally block
```