**OOPM NOTES**

What is Procedural Programming?

It is a programming paradigm derived from structured programming. It is based on the concept of procedure call. Procedures, also known as routines, subroutines, or functions, simply contain a series of computational steps to be carried out. Eg. C, C++, PASCAL

What is Object Oriented Programming?

Object-oriented programming (OOP) is a programming language model in which programs are organized around data, or objects, rather than functions and logic. program is composed of a collection of objects that communicate with each other. Example: Java, Python, JavaScript

Difference between C, C++, Java?

**Points:**

Programming Paradigm

Pointers

Preprocessor directives

Header files

Memory Allocation

DB connectivity

Code Translation (Compiled/Interpreted)

Complex Data Types

String Type

Exception handling yes yes yes

Operator Overloading no yes no

Multithreading & interfaces no no yes

Inheritance no yes yes (no multiple inheritance)

What is Java?

Java is a programming language and a platform. It is a high level, robust, object-oriented and secure programming language. Platform: Any hardware or software environment in which a program runs, is known as a platform.

Features of Java?

Simple (easy to learn, simple syntax)

Object Oriented (• Object, Class • Inheritance • Polymorphism • Abstraction • Encapsulation)

Secure & Robust (No explicit pointer, runs inside virtual machine, exception handling, strong memory management)

Platform independent

interpreted

Portable

High Performance (faster than other interpreted languages because Java bytecode is "close" to native code. slower than a compiled language because java is an interpreted language)

Multithreaded (deal with many tasks at once by defining multiple threads)

Architecture neutral (no implementation-based features, size of primitive types is fixed)

Distributed (facilitates users to create distributed applications)

Dynamic (supports dynamic loading of classes. Classes are loaded only when they are needed)

What is an Object?

An object is an instance of a class. It means that class is a template/blueprint from which objects are created. Object is both a physical as well as logical entity. They have three characteristics:

State: represents data of an object (class variables)

Behaviour: Represents functionality of the object (methods)

Identity: Object identity is typically a unique id which is not shown to the user but used internally by the JVM.

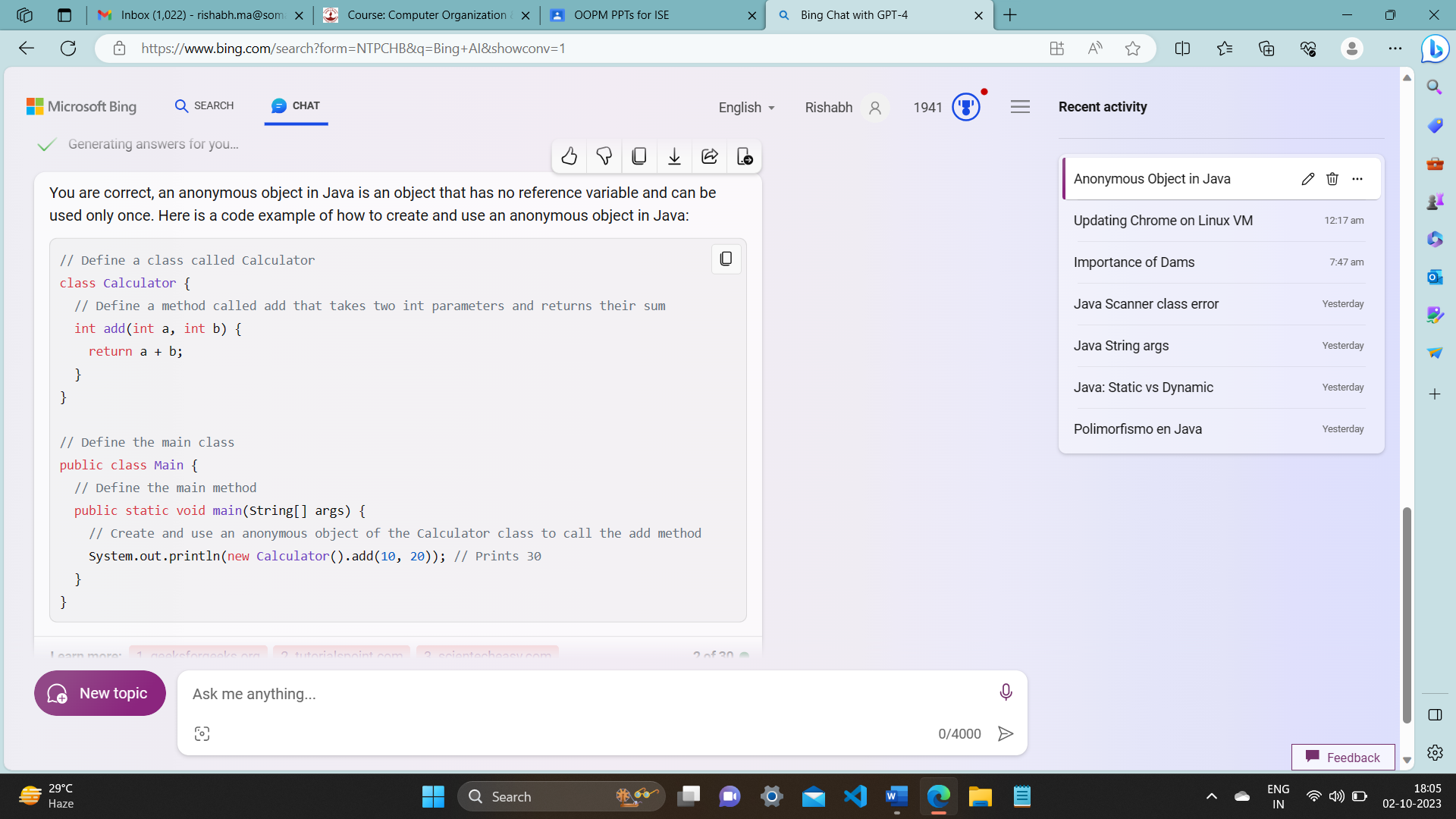
Objects r created using:

new keyword- new keyword is used to allocate memory at runtime.

newInstance() method

clone() method

What is an anonymous object?



What is a Class?

A class defines a group of objects having common properties. It is a template/blueprint from which objects are created. It is a logical entity. Classes can contain variables, methods, constructors, nested classes and interface.

The 4 basic properties of OOP

**Inheritance:**

Inheritance is a mechanism in which a child class can acquire all the properties and behaviour of the parent class. The class that is inherited from is called the "parent" or "superclass," and the class that inherits is called the "child" or "subclass."

It is useful because it helps code reusability and thus prevents the need of rewriting code. You can create new classes based on existing classes so we can reuse the methods and fields of the parent class while defining new methods and fields in the child class.

Inheritance creates a hierarchical relationship between classes.

Inheritance represents IS-A relationship or parent-child relationship( dog IS-A animal)

**Polymorphism:**

Polymorphism is a concept by which a single action can be performed in multiple ways. It is done using method overloading and method overriding.

There are two types of polymorphism in java, compile time polymorphism and runtime polymorphism.

Compile time polymorphism (static binding) is achieved by method overloading. In method overloading, multiple methods with the same name but different parameter lists are defined within the same class. The parameter lists may differ by number of arguments or types of arguments or both.

class Calculator

{

int add (int a, int b)

return a + b;

double add (double a, double b, double c)

return a + b + c;

}

Runtime polymorphism (dynamic binding) is achieved through method overriding. In method overriding, a subclass provides a specific implementation for a method that is already defined in its superclass. The decision about which method to call is made at runtime based on the actual type of the object.

class Animal {

void makeSound() {

System.out.println("Animal makes a sound");

}

}

class Dog extends Animal {

@override

void makeSound() {

System.out.println("Dog barks");

}

}

Dynamic Method Dispatch:

The process by which the correct method is called at runtime is known as dynamic method dispatch. It allows a subclass to provide a specific implementation of a method while still adhering to the method signature in the superclass.

Animal myPet = new Dog(); myPet.makeSound();

// Calls the makeSound method of Dog at runtime

Here, the reference variable **myPet** is of type **Animal**, but it refers to an instance of **Dog**. The method **makeSound** invoked is determined at runtime based on the actual type of the object (runtime polymorphism).

**Abstraction:**

Abstraction is a way of hiding the implementation details from the user and providing only a well-defined interface, i.e. showing only the functionality and hiding the internal details. Example: sending SMS where you type the text and send the message. You don't know the internal processing about the message delivery.

**Encapsulation**:

Encapsulation in Java is a process of wrapping data (attributes) and methods (functions) together in a single unit. In java this is achieved through classes. Example, a capsule which is mixed of several medicines.

How is java platform independent?

Java is often referred to as platform-independent because of its "Write Once, Run Anywhere" (WORA) capability.

There is a 2-step code translation process.

Source code 🡪 machine independent byte code by javac compiler. It can be executed on any platform (Windows, Linux, Mac etc)

Byte code 🡪 machine code by interpreter i.e., JVM

What is multithreading in java?

Multithreading in Java refers to the capability of a program to execute multiple threads concurrently within a single process. A thread is like a separate program, executing concurrently. We can write Java programs that deal with many tasks at once by defining multiple threads. The main advantage of multi-threading is that it doesn't occupy memory for each thread. It shares a common memory area. Threads are important for multi-media, Web applications, etc.

How is java robust? (stable and strong)

It has strong memory management.

It has no explicit pointers.

Java Programs run inside a virtual machine.

There is an exception handling mechanism in java.

How is java architecture neutral?  
Java is architecture neutral because there are no implementation dependent features, for example, the size of primitive types is fixed.

How is java high performance?

Java is faster than other traditional interpreted programming languages because Java bytecode is "close" to native code. Java is an interpreted language that is why it is a little slower than compiled languages (e.g. C++).

Java is a dynamic language. Explain

Java is a dynamic language. It supports dynamic loading of classes. It means classes are loaded on demand.Java supports dynamic compilation and automatic memory management (garbage collection)

What are the ways to read input in Java?  
Command Line Interpreter- args.length – without bracket

Buffered reader Class

Scanner Class

What is a static/class variable ? What is instance variable ?

When a variable is declared with keyword static it is called a class variable. All instances of the class share the same variable. a class variable can be accessed without the need of creating an object which makes it memory efficient. If one object increases it by one and another by 1 then finally the count is 2. If count was an instance variable, then each object of the class would have its own copy of the variable and it would not be shared by all objects of the same class. In that case, the value of count would be 1 for each object, because it would only be incremented by one when that object is created.

A class variable is declared inside the class but outside any method with the keyword static, and it is initialized when the class is loaded. A class variable can have only one value for all objects of the same class. A class variable can be accessed by using the class An instance variable is declared inside the class but outside any method, and it is initialized when an object of the class is created. An instance variable can have different values for different objects of the same class. An instance variable can be accessed directly by the object name

‘this’ keyword in java

The this keyword in Java is a reference variable that refers to the current instance of a class. Using this you can refer the members of a class such as constructors, variables and methods. It can be used for various purposes, especially when there is a name conflict between the local variables and the instance variables

1. this keyword can be used to refer current class instance

2. this keyword can be used to invoke current class constructor.

3. this keyword can be used to invoke current class method (implicitly)

4. this can be passed as an argument in the method call.

5. this can be passed as argument in the constructor call.

6. this keyword can also be used to return the current class instance.

What is a method?

A method in java is a collection of statements grouped together to perform a specific operation. Modifier is optional. There are two ways in which a method is called i.e. method returns a value or returning nothing

Syntax:

Modifier returnType nameOfMethod (Parameter List)

{ // body }

What is a static method in java?

A static method belongs to the class rather than an object of the class. Static method can be called without the need of creating an object of the class. Static method can access the static data member and can change the value of it. The static method cannot use non static data member or call a non-static method. You cannot use keywords “this” and “super”. If you wish to call non static method of same class then you have to create object. If you wish to call static method of another class then you have to write other class name while calling static method. If you wish to call non static method of another class then you have to create object of other class.

What is static block?

Is used to initialize the static data member. It is executed before main method at the time of class loading. So this is one of the way to execute a program without main() method.

class A2

{

static{System.out.println("static block is invoked");

}

public static void main(String args[])

{

System.out.println("Hello main");

}

OUTPUT:

static block is invoked

Hello main

**Operators in java**

Unary Operator +, -, ++, --

Arithmetic Operator +, -, /, \*, %

Relational Operator >, >=, <, <=, ==, !=

Logical Operator &&, ||, !

Assignment Operator =, +=, -=, \*=, /=, %=

Shift Operator **<<** (left shift), **>>** (right shift)

Bitwise Operator &, |(OR), ^(XOR)

Ternary Operator ?:

What are access specifiers/modifiers in java?

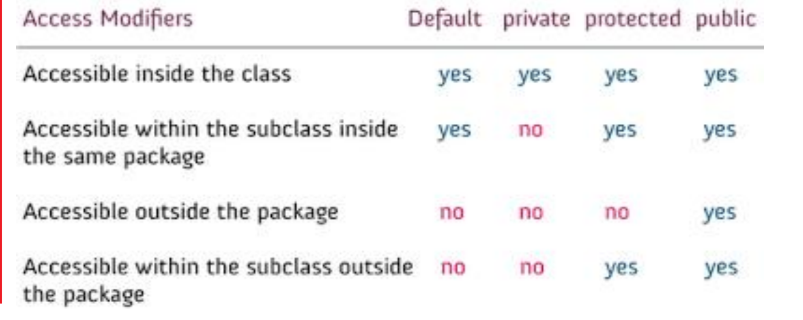
The access modifiers in Java specifies the accessibility of a field, method, constructor, interface or class. We can change the access level of fields, constructors, methods, interface and class by applying the access modifier on it.

**Public**: Class, Method, Constructor or Interface which are public can be accessed from any other class in the java universe i.e., both within package and outside package. (However, if the public class we are trying to access is in a different package, then the public class still need to be imported. Because of class inheritance, all public methods and variables of a class are inherited by its subclasses)

**Protected**: The access level of methods, variables and constructors declared protected is all classes within the package and subclasses outside the package. Cannot be applied to class and interface. Methods and fields in an interface cannot be declared protected. Protected access gives the subclass a chance to use the method or variable, while preventing a nonrelated class from trying to use it.

**Default**: Default access modifier is applied when we do not explicitly declare an access modifier. The variable or method which has default access level can be accessed from any class within the same package.

**Private**: Methods, Variables and Constructors which are private can only be accessed from within the class itself. Classes and interfaces cannot be private. Private access modifier is the most restrictive access level. Using the private modifier is the main way that an object encapsulates itself and hide data from the outside world.



What is a constructor?

A constructor initializes an object when it is created.

It has the same name as the class name and has no return type.

A constructor is used to give initial values to the instance variables defined by the class and perform any other startup procedures required.

All classes have constructors, whether you define one or not as java automatically provides a default constructor that initializes all the variables to 0.

What are the types of constructors in java?

There are two types of constructors:

Default Constructor:

Default constructor provides the default values to the object like 0, NULL etc. If no constructor is defined for a class, default constructor is used to initialize the object.

Parameterised Constructor:

It is a constructor that has parameters. It is used to provide different values to different objects.

\*There is no copy constructor in java. But, we can copy the values of one object to another :

By constructor

By assigning the values of one object into another

By clone() method of Object class

What is constructor overloading?

Constructor overloading in Java is a technique that allows a class to have more than one constructor with different parameters. The compiler distinguishes between the overloaded constructors by looking at the number, type and order of the parameters.

Here is an example of a class called Rectangle that has three overloaded constructors:

// Define a class called Rectangle

class Rectangle {

// Declare two instance variables: length and width

int length;

int width;

// Define a constructor that takes no parameters and sets the length and width to zero

Rectangle() {

length = 0;

width = 0;

}

// Define a constructor that takes one parameter and sets the length and width to that value

Rectangle(int side) {

length = side;

width = side;

}

// Define a constructor that takes two parameters and sets the length and width to those values

Rectangle(int length, int width) {

this.length = length;

this.width = width;

}

// Define a method that returns the area of the rectangle

int area() {

return length \* width;

}

}

// Create an object of the Rectangle class using the default constructor

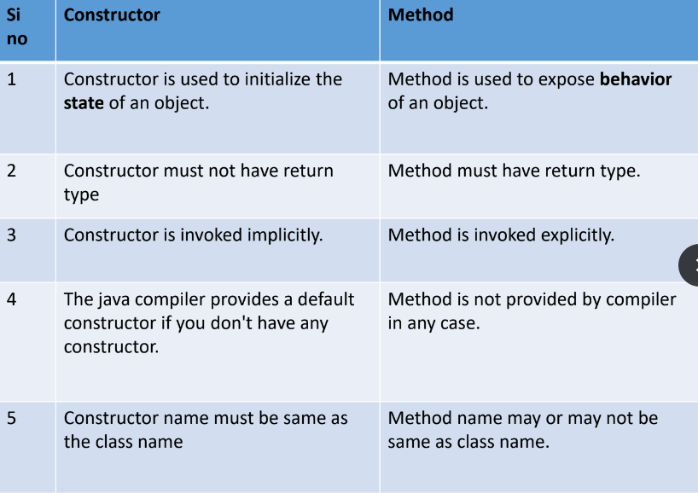
Rectangle r1 = new Rectangle();

// Create an object of the Rectangle class using the constructor with one parameter

Rectangle r2 = new Rectangle(5);

// Create an object of the Rectangle class using the constructor with two parameters

Rectangle r3 = new Rectangle(3, 4);



What is a destructor?

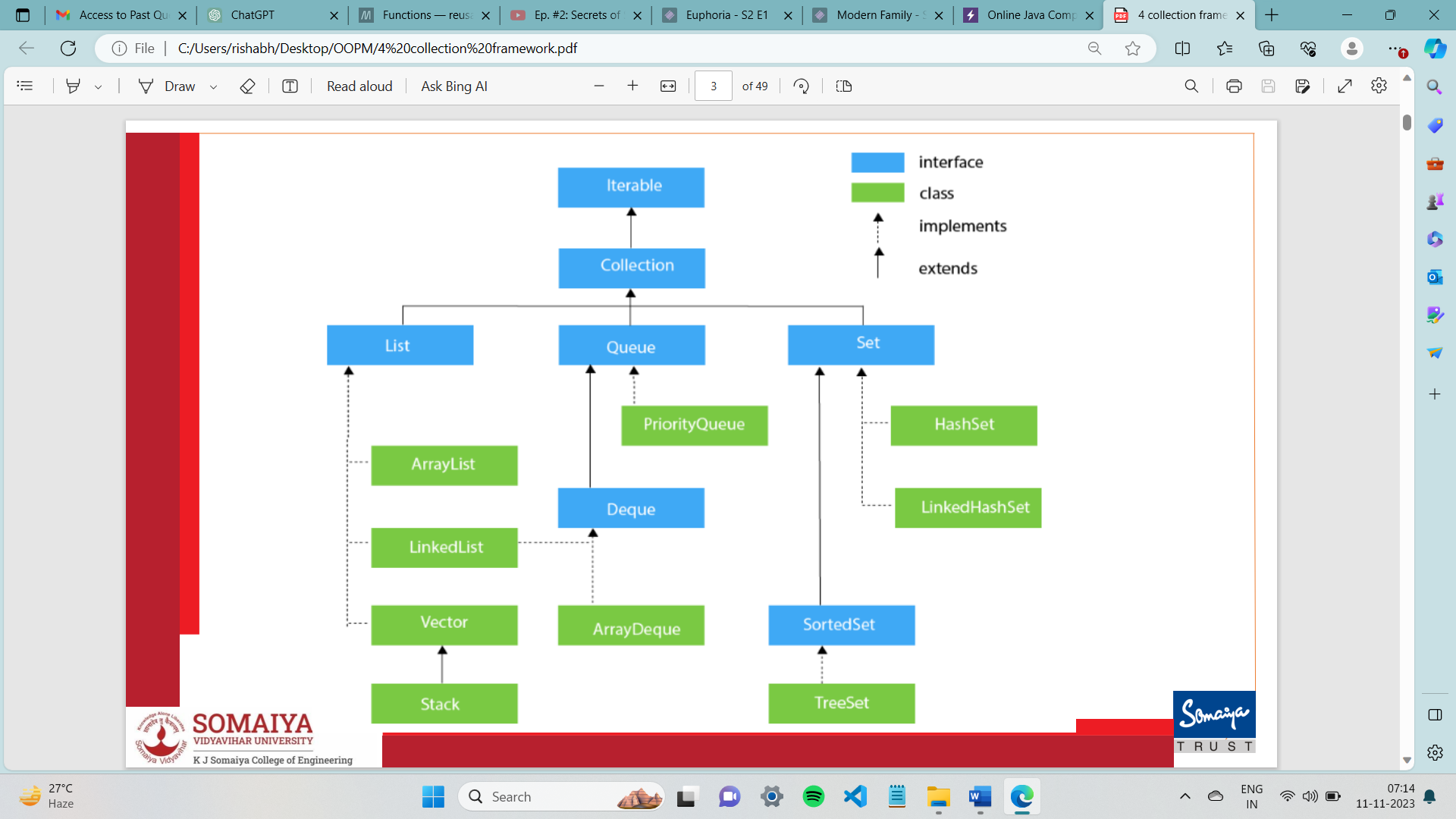
A destructor is a special member function of a class that is executed whenever object goes out of scope or whenever the delete operation is applied to the pointer of an object. It has the same name as the class name and has the tilde (~) as a prefix. It has no arguments or return types. Destructor can be very useful for releasing resources before coming out of the program like closing files, releasing memories etc

What is the finalise method?

In java the finalise method is equivalent to the c++ concept of destructor. When the job of an object is over, or to say, the object is no more used in the program, the object is known as garbage. The process of removing the object from a running program is known as garbage collection. In Java, memory is managed automatically by the garbage collector, and objects are automatically deallocated when they are no longer reachable. The finalize () method is a method of the **Object** class, and it gets called by the garbage collector before an object is reclaimed. finalize () method can be best utilized by the programmer to close the I/O streams.

What is Collection Framework?

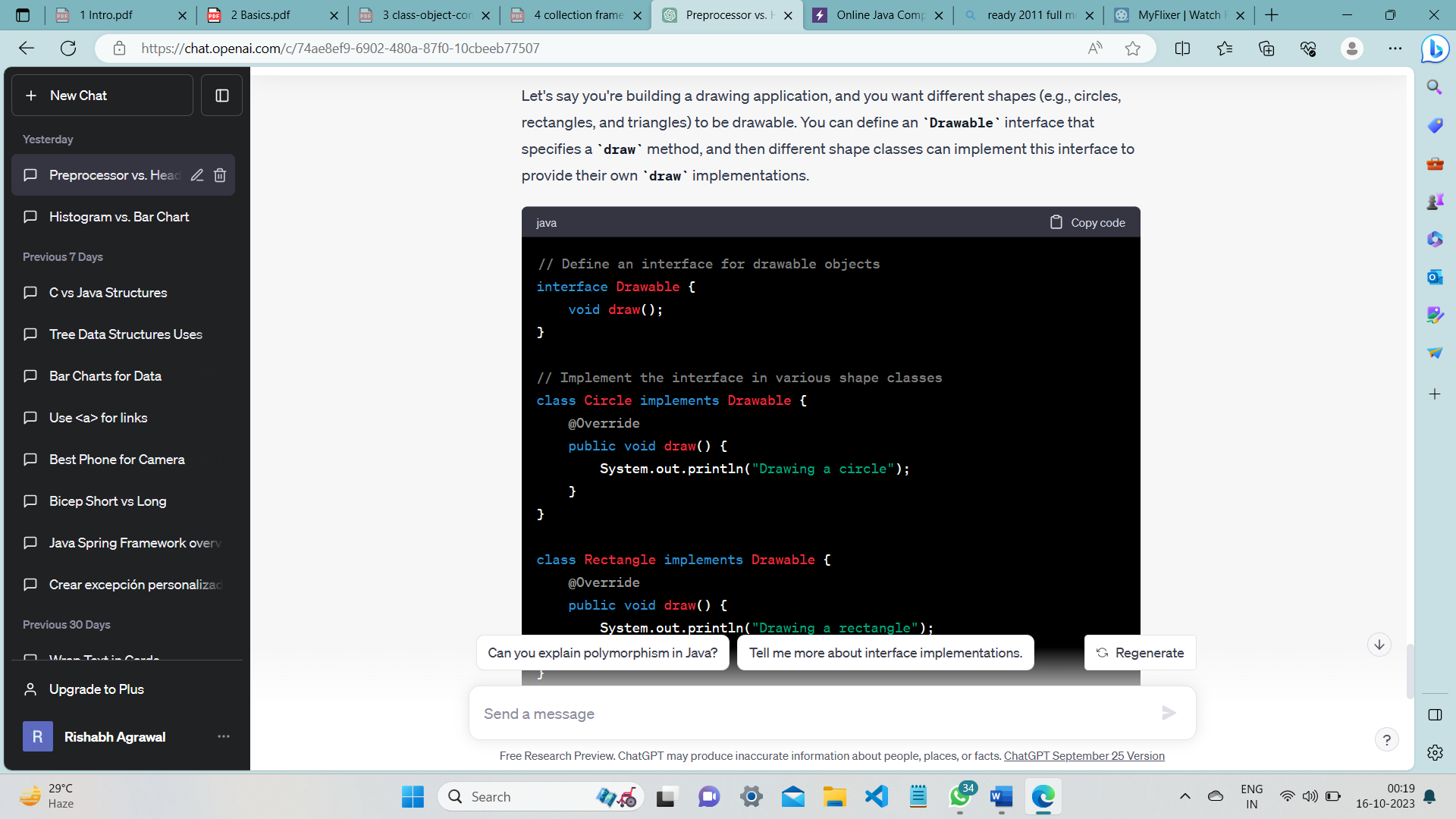
The Java Collections Framework is a set of classes and interfaces in Java that provide an architecture to store, organize, and manipulate groups of objects. It helps to achieve all the operations that you perform on a data such as searching, sorting, insertion, manipulation, and deletion. It provides a unified and consistent interface for working with different types of collections, making it easier for developers to work with data structures without having to worry about the underlying implementation details. Java Collection means a unit of objects.



What is an interface in java?

In Java, an interface is a blueprint of a class that defines a set of abstract methods (methods without a body) that any class implementing the interface must provide concrete implementations for. It is similar to an abstract class but ALL methods in an interface must be abstract. They are declared using interface keyword and can be implemented by other classes using the implement keyword. They are used to achieve 100% abstraction. A class can extend only 1 class but it can implement multiple interfaces. You cannot create objects directly from an interface. Objects are created from classes that implement the interface.

NOTE: if an abstract class implements an interface, it NEED NOT implement all methods defined in the interface. HOWEVER, each concrete (not abstract) subclass MUST implement the methods defined in the interface. If the concrete subclass does not provide an implementation for any of the methods declared in the interface or inherited from the abstract class, it must be declared as an abstract class itself. Whenever a class implements an interface, they must provide definitions for all the methods in that interface. Interfaces cannot contain instance variables (i.e. Only public, static and final variables are allowed).

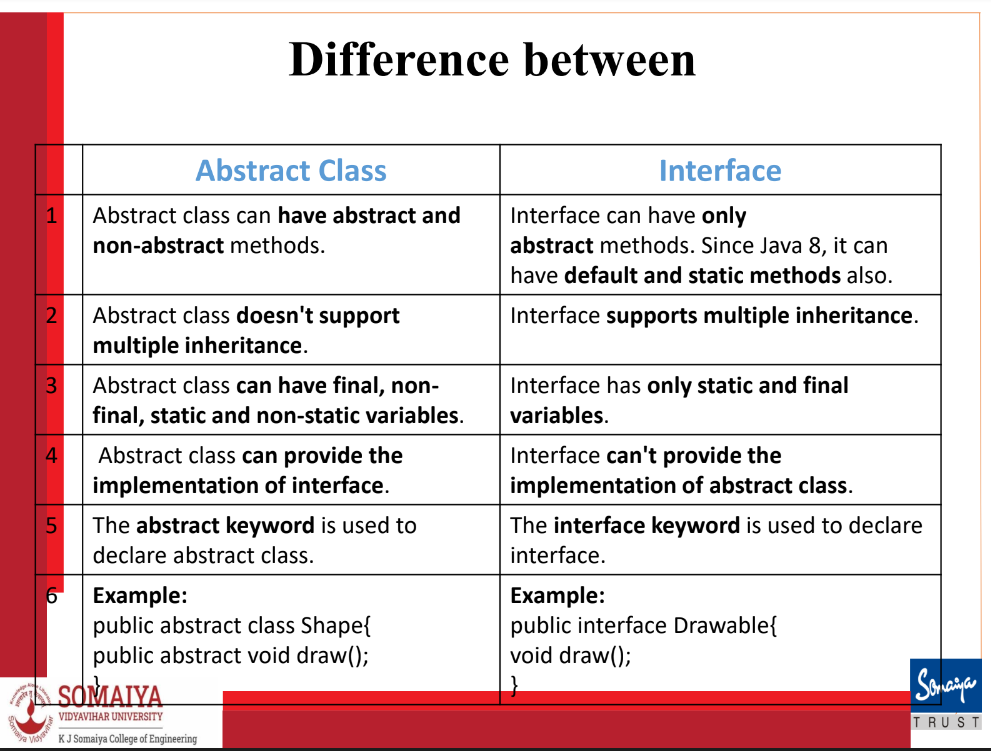


Why do we use interface?

* They are used to achieve 100% abstraction.
* They can be used to achieve effects of multiple inheritance which is not allowed in Java.
* Interfaces define a common set of methods that multiple classes can adhere to.

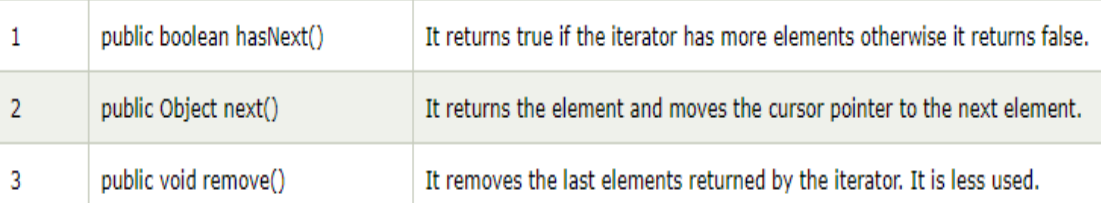
Can one interface inherit another interface?

Yes, one interface can inherit another by use of the keyword extends. The syntax is the same as for inheriting classes. When a class implements an interface that inherits another interface, it must provide implementations for all methods defined within the interface inheritance chain.



What is iterator interface?

Iterator interface provides the facility of iterating the elements in a forward direction only.



What is collection interface?

Interface which is implemented by all the classes in the collection framework. It declares the methods that every collection will have. Collection interface builds the foundation on which the collection framework depends.

What is List Interface?

It can have duplicate values.

To instantiate the List interface, we must use :

List <data-type> list1= new ArrayList();

List <data-type> list2 = new LinkedList();

List <data-type> list3 = new Vector();

List <data-type> list4 = new Stack();

Which is correct from below 2 ?

List <data-type> list2 = new LinkedList();

ArrayList <data-type> list2 = new ArrayList();

Both of the declarations you provided are correct, but they are used in slightly different contexts:

1. Using the LinkedList Type Explicitly:

In this declaration, you are explicitly stating that list2 is of type LinkedList. While this is syntactically correct and might be required if you need to use specific methods or properties of LinkedList, it's generally recommended to use the more flexible list interface as the type.

1. Using the List Interface:

In this declaration, you are using the List interface as the type. This is often preferred because it allows you to switch the underlying implementation (e.g., from LinkedList to ArrayList) without changing the rest of your code. This is known as coding to the interface, and it promotes flexibility and easier maintenance. list2 = new ArrayList<>(); // Change the implementation to ArrayList

So, in general, the second declaration is more flexible and adheres to good coding practices. It's often recommended to use interfaces when declaring variables to allow for easier changes in the future. The underlying implementation (LinkedList in this case) can be swapped without affecting the rest of the code as long as you only use methods declared in the List interface.

What are arrays in Java?

Array is a collection of elements that have contiguous memory location. Java array is an object the contains elements of homogenous data type. We can store only fixed number of elements in a java array. They are fast since they are low level.

There are two types of array.

* Single Dimensional Array

int[] numbers = {1, 2, 3, 4, 5};

int[] numbers = new int[5];

* Multidimensional Array

int[][] matrix = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};

int[][] matrix = new int[3][3];

What is Jagged Array?

It is a new feature supported by Java, where column size varies, i.e. each row may have varying length.

10 20 30

11 22 22 33 44

77 88

int twoD[][] = new int[3][];

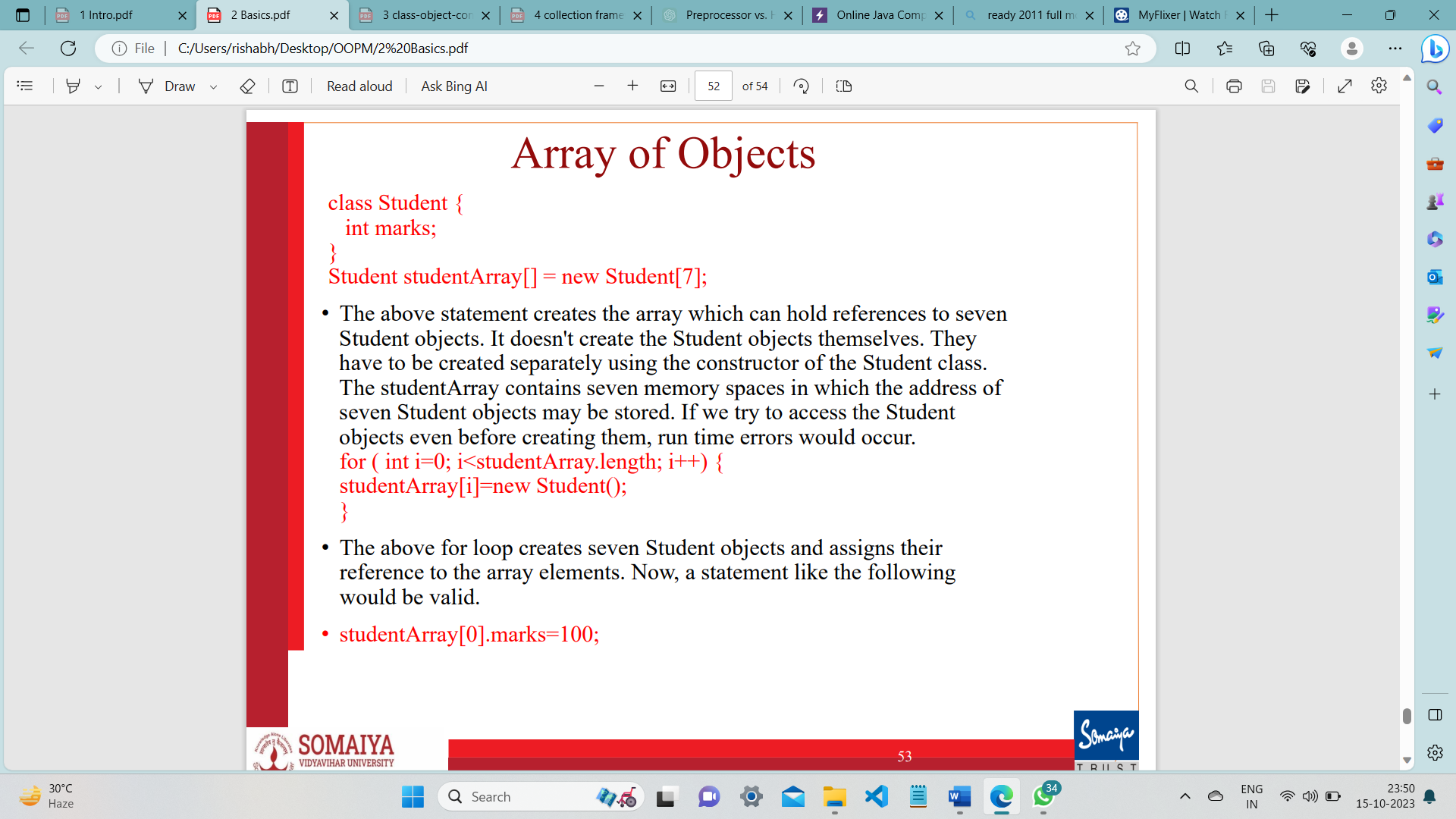
twoD[0] = new int[3];

twoD[1] = new int[5];

twoD[2] = new int[2];

What is array of objects?

You can create an array of objects to hold instances of a class. Like regular arrays, arrays of objects have a fixed size once created. It allows direct access to elements based on index.



What is ArrayList ?

Java ArrayList class uses a dynamic array for storing the elements (there is no size

limit)

We can add or remove elements anytime, much more flexible than the traditional array.

It is found in the java.util package.

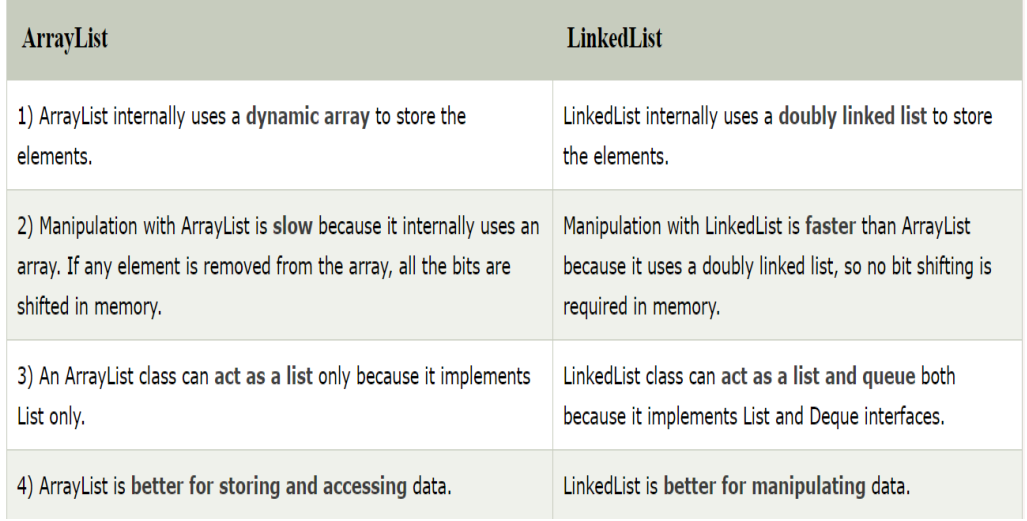
The ArrayList in Java can have the duplicate elements also.

It implements the List interface so we can use all the methods of List interface here.

ArrayList increments 50% of the current array size if the number of elements exceeds its capacity

What is LinkedList?

• Java LinkedList class uses a doubly linked list to store the elements.

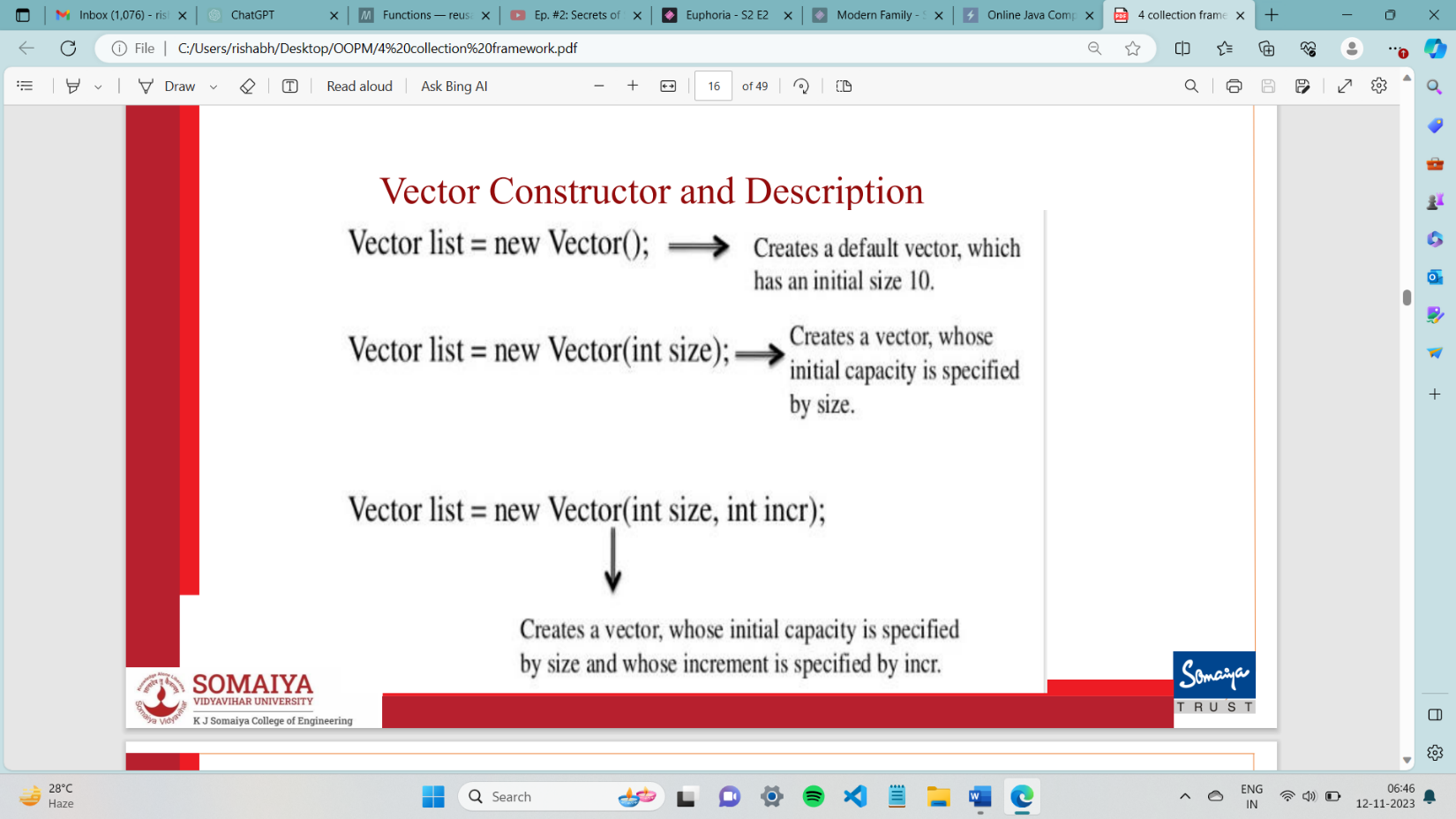


What are vectors ?

Vectors are a dynamic array of objects which can store heterogenous objects. They are useful when you don’t know the exact size of array in advance or you just need one that can change sizes over the lifetime of a program as they can grow and shrink as necessary. We cannot store primitive data types in array they must be converted to objects, as vector can only store objects. A vector has an initial capacity, if this capacity is reached then size of vector automatically increases. Vectors are synchronized. Its defined in java.util package. To traverse elements of a vector class we use Enumeration interface. This class is a member of the Java Collections Framework.

• This default initial capacity of vectors are 10. The default increment is double the size.

• Each vector tries to optimize storage management by maintaining a capacity and a capacityIncrement arguments.



\*\*V has to be caps.

Differences between a Vector and an Array/array of objects

* A vector is a dynamic array, whose size can be increased, where as an array size cannot be changed.
* A vector is a class where as an array is not.
* Vectors can store any type of objects, where as an array can store only homogeneous values.
* Vector is part of collection framework
* Vector is synchronized

Advantages of Arrays:

* Arrays supports efficient random access to the members.
* They are more appropriate for storing fixed number of elements

Disadvantages of Arrays:

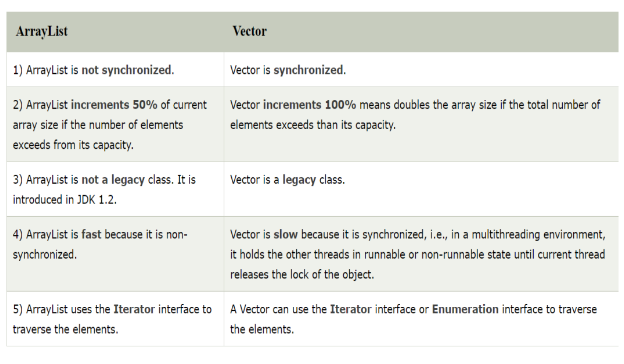
* Dynamic creation of arrays is not possible
* Multiple data types cannot be stored

Advantages of Vector:

* Size of the vector can be changed
* Multiple objects can be stored

Disadvantages of Vector:

* A vector is an object, memory consumption is more.
* They are more appropriate for storing uncertain number of elements



When to used what?

* Use arrays when direct access by index is important and you do not need to insert or delete elements frequently.
* Use Vector if you need a thread-safe dynamic array.
* Use Array List for a dynamic array without the need for synchronization.
* Use LinkedList when frequent insertions or deletions are required and you do not need to access elements by their index frequently.

Methods unique to collection interface

1. **boolean add(E element)**: Adds the specified element to the collection.
2. **boolean addAll(Collection<? extends E> c)**: Adds all elements from the specified collection to the collection.
3. **boolean remove(Object o)**: Removes the first occurrence of the specified element from the collection.
4. **boolean removeAll(Collection<?> c)**: Removes all elements from the collection that are contained in the specified collection.
5. **boolean retainAll(Collection<?> c)**: Retains only the elements in the collection that are contained in the specified collection.
6. **void clear()**: Removes all elements from the collection.
7. **boolean contains(Object element)**: Returns **true** if the collection contains the specified element.
8. **boolean containsAll(Collection<?> c)**: Returns **true** if the collection contains all elements in the specified collection.
9. **int size()**: Returns the number of elements in the collection.
10. **boolean isEmpty()**: Returns **true** if the collection contains no elements.
11. **Object[]name toArray()**: Returns an array containing all of the elements in the collection.
12. **boolean equals(Object o)**: Compares the specified object with this collection for equality.
13. **Iterator<E> iterator()**: Returns an iterator over the elements in the collection.

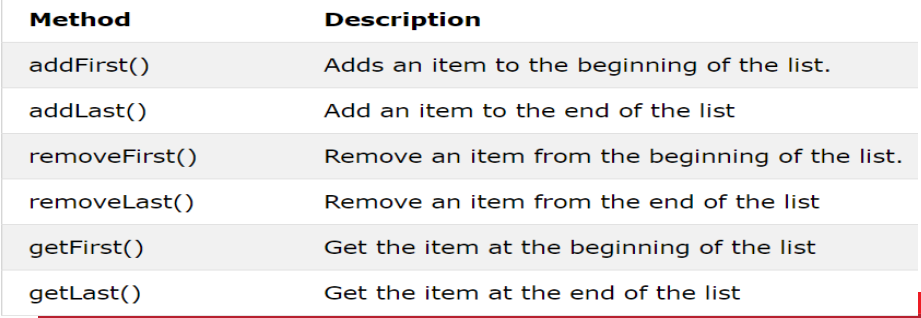
Methods unique to list interface

1. **void add(int index, E element)**: Inserts the specified element at the specified position in the list.
2. **boolean addAll(int index, Collection<? extends E> c)**: Inserts all elements in the specified collection into the list at the specified position.
3. **E remove(int index)**: Removes the element at the specified position in the list.
4. **E get(int index)**: Returns the element at the specified position in the list.
5. **E set(int index, E element)**: Replaces the element at the specified position in the list with the specified element.
6. **int indexOf(Object o)**: Returns the index of the first occurrence of the specified element in the list.
7. **int lastIndexOf(Object o)**: Returns the index of the last occurrence of the specified element in the list.
8. **List<E>name subList(int fromIndex, int toIndex)**: Returns a view of the portion of the list between the specified **fromIndex** (inclusive) and **toIndex** (exclusive).
9. **ListIterator<E> listIterator()**: Returns a list iterator over the elements in the list.
10. **ListIterator<E> listIterator(int index)**: Returns a list iterator over the elements in the list, starting at the specified position.

Methods unique to vector class

1. void addElement(Object Element) :Adds the element to the end of this vector, increasing its size by one.
2. void removeElementAt(int index) : Removes the element at the specified position in this vector. Shifts any subsequent elements to the left (subtracts one from their indices).
3. void removeAllElements() : Removes all components from this vector and sets its size to zero
4. E elementAt(int index) : Returns the element at the specified index in the vector.
5. void insertElementAt(Object element, int index) : Inserts the specified element at the specified position in the vector, shifting the current elements at that position and beyond to the right.
6. void setElementAt(Object element, int index) : Sets the element at the specified index of this vector to be the specified element.
7. E firstElement(): Returns the first element in the vector.
8. object lastElement() : Returns the last element in the vector.
9. int capacity() : Returns the current capacity of the StringBuilder or StringBuffer instance. The capacity is the maximum number of characters the instance can hold without reallocating its internal buffer.
10. void setSize(int newSize) : This method is used to set the size of the vector. If the current size is greater than the specified size, the vector is truncated. If the current size is less than the specified size, new **null** elements are added at the end of the vector.
11. void trimToSize() : Reduces the capacity of this Vector to be its current size.
12. void copyInto(Object[] Array) : used to copy the components of the vector into the specified array. The method copies elements from the beginning of the vector to beginning the specified array.it replaces any pre-existing elements if they come in the way. If the specified array is smaller than the size of the vector, only the elements that fit into the array are copied.

Methods unique to LinkedList



What are wrapper classes?

Java is an object-oriented language and can view everything as an object. A Wrapper class in Java is a class whose object wraps or contains primitive data types. They are used to convert primitive data types to objects. It is useful because classes in the collection framework only allow objects as data type and not primitive data types, so Wrapper classes help us overcome this problem.

EXAMPLE  
int k = 100; //The int data type k is converted into an object, it1 using Integer class.

Autoboxing /wrapping

Integer it1 = new Integer(k);

Unboxing/Unwrapping

int m = it1.intValue();

Basic Type- Wrapper

boolean- Boolean

char- Character

int- Integer

long- Long

float- Float

double- Double

What is autoboxing and unboxing?

The process of automatically converting a primitive data type into the object of its corresponding wrapper class is called autoboxing.

The process of automatically converting a Wrapper Class Object into its corresponding primitive data type is called unboxing.

//initializing using method

class Rectangle{

int length;

int width;

void insert(int l,int w){

length=l;

width=w;

}

public static void main(String args[]){

Rectangle r1=new Rectangle();

r2=new Rectangle();

r1.insert(11,5);

r2.insert(3,15);

// initialize using constructor

class Rectangle {

int length;

int width;

Rectangle(int l, int w) {

// Assign the parameters to the instance variables

length = l;

width = w;

}

// main method

public static void main(String args[]) {

// Create two objects of the Rectangle class using the constructor

Rectangle r1 = new Rectangle(11, 5);

Rectangle r2 = new Rectangle(3, 15);

What is a String class?

String class provides many operations for string manipulations like constructors, utility, comparisons, conversions, etc.

String objects are read-only, i.e. immutable.

Declaration:

String stringName;

stringName = new String (“string value”);

What is a StringBuffer class?

All the operations of String class can be performed (Refer above).

Unlike the String class, StringBuffer class is mutable (changeable).

StringBuffer class is used in operations where the string has to be modified.

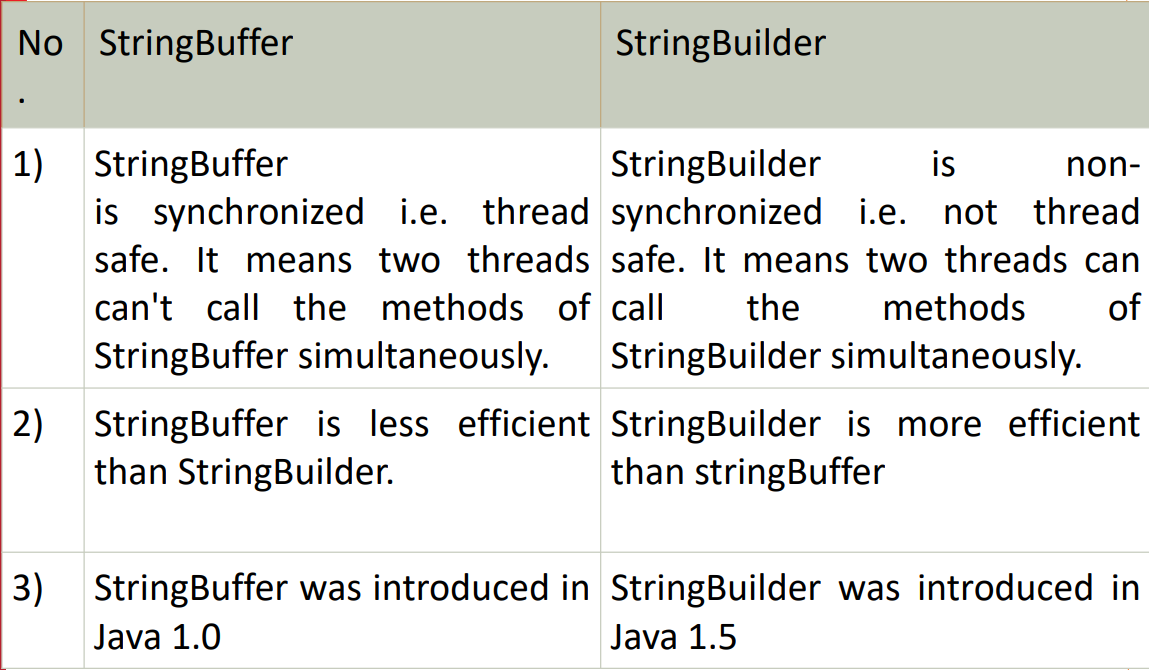
StringBuffer is synchronized i.e. thread safe.

What is a StringBuilder class?

The StringBuilder in Java represents a mutable sequence of characters and provides an alternative to String Class, as it creates a mutable sequence of characters.

The function of StringBuilder is very much similar to the StringBuffer class, However the StringBuilder class differs from the StringBuffer class on the basis of synchronization. It is non - synchronized.

* Use **String** when you need immutable string(string is not expected to change frequently.)
* Use **StringBuffer** when you need a mutable string and thread safety is a concern.
* Use **StringBuilder** when you need a mutable string and where thread safety is not critical.



STRING CLASS METHODS

int length() :Returns the length of the string.

charAt(int index) :Returns the character at the specified index

int compareTo( String anotherString) : Compares the Strings.

int compareToIgnoreCase( String anotherString) :Compare the Strings.

String replace(char oldChar, char newChar) : Returns a new string with all instances of the oldChar replaced with newChar.

trim() : Trims leading and trailing white spaces.

String toLowerCase(): converts to lowercase

String toUpperCase(): converts to uppercase

boolean equals(String anotherString): Returns true if Strings are equal

bolean equalsIngnoreCase(String s1): Returns true if Strings are equal

String concat(String S1): Returns concatenated string

String substring(int beginIndex, int endIndex): Returns substring starting from beginIndex character to endIndex character(not including endIndex character)

String substring(int beginIndex): Returns substring from beginIndex character to end

Char[] toCharArray(): Convert this string to character array.

STRINGBUFFER CLASS METHODS

int length(): Returns the length of the buffer

synchronized void setCharAt(int index, char ch): Replaces the character at the specified position

s1.setLength(int n): Truncates or extends the buffer. If n<s1.length(), s1 is truncated else zeroes are added to s1

StringBuffer append(String str): Appends the string to this string buffer.

StringBuffer append(datatype x): Appends the string representation of the argument to this string buffer.

STRINGBUILDER CLASS METHODS

• StringBuilder append(X x): This method appends the string representation of the X type argument to the sequence.

• int capacity(): This method returns the current capacity.

• char charAt(int index): This method returns the char value in this sequence at the specified index.

• StringBuilder delete(int start, int end): This method removes the characters in a substring of this sequence.

• void ensureCapacity(int minimumCapacity): This method ensures that the capacity is at least equal to the specified minimum.

• void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin): This method characters are copied from this sequence into the destination character array dst.

• int indexOf(): This method returns the index within this string of the first occurrence of the specified substring.

• StringBuilder insert(int offset, boolean b): This method inserts the string representation of the booalternatelean argument into this sequence.

• StringBuilder insert(): This method inserts the string representation of the char argument into this sequence.

• int lastIndexOf(): This method returns the index within this string of the last occurrence of the specified substring.

• int length(): This method returns the length (character count).

• StringBuilder replace(int start, int end, String str): This method replaces the characters in a substring of this sequence with characters in the specified String.

• StringBuilder reverse(): This method causes this character sequence to be replaced by the reverse of the sequence.

• String substring(): This method returns a new String that contains a subsequence of characters currently contained in this character sequence.

• String toString(): This method returns a string representing the data in this sequence

What is toString() Method?

• toString() method is a special method that can be defined in any class.

• This method should return a String argument.

• When an object is used in print statement or when one tries concatenation operation with Strings, this method gets invoked automatically.

Example:

class Circle {

double x, y, r;

public Circle(double centreX, double centreY, double radius) {

x = centreX;

y = centreY;

r = radius;

}

public String toString() {

String s = "I am a Circle with centre [" + x + "," + y + "] and radius [" + r + "]";

return s;

}

}

public class Main {

public static void main(String[] args) {

Circle c = new Circle(10, 20, 30);

System.out.println(c);

// I am a Circle with centre [10.0,20.0] and radius [30.0]

}

}

What is inheritance in Java?

Inheritance is a mechanism in which a child object/class acquires all the properties of the parent object/class.

It is useful because it helps code reusability and thus prevents the need of rewriting code. You can create new classes based on existing classes so we can reuse the methods and fields of the parent class while defining new methods and fields in the child class.

Inheritance represents IS-A relationship or parent-child relationship.

Java Classes can be reused by the help of inheritance.

Inheritance is used for method overriding.

A class in java can either EXTEND another class or IMPLEMENT an interface.

Example code:

class Employee {

    static int salary = 10000;

}

public class Programmer extends Employee {

    static int bonus = 5000;

    public static void main(String[] args)

    {

        System.out.println("Salary is " + salary);

        System.out.println("Bonus is " + bonus);

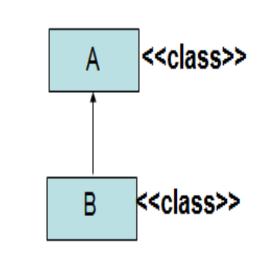
    }

}

Types of Inheritance in Java

1. Single Inheritance:

Type of inheritance in which one class extends only one other class is called Single Inheritance.



2. Multilevel Inheritance:

In Multilevel Inheritance one class inherits from a derived class which itself inherits from another class. Therefore, a derived class becomes a superclass for the new class. This forms a chain of inheritance.

Example Code:

class Human {

    static String name = "Rahil";

}

class Employee extends Human{

    static int salary = 10000;

}

public class Programmer extends Employee {

    static int bonus = 5000;

    public static void main(String[] args)

    {

        System.out.println("Name is " + name);

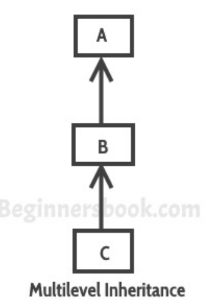
        System.out.println("Salary is " + salary);

        System.out.println("Bonus is " + bonus);

    }

}

Diagram:



1. Hierarchical inheritance:

In hierarchical inheritance, multiple subclasses inherit from a single superclass.

Example Code:

class Employee{

    static int salary = 10000;

}

class Manager extends Employee {

    static int bonus = 10000;

}

class CEO extends Employee {

    static int bonus = 100000;

}

public class Programmer extends Employee {

    static int bonus = 5000;

    public static void main(String[] args)

    {

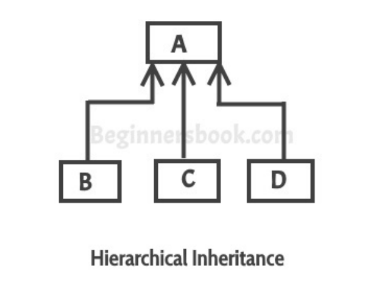
        System.out.println("Salary is " + salary);

        System.out.println("Bonus is " + bonus);

    }

}

Diagram:



1. Multiple Inheritance

In multiple inheritance one subclass tries to inherit more than one superclass.

MULTIPLE INHERITANCE IS NOT SUPPORTED IN JAVA.

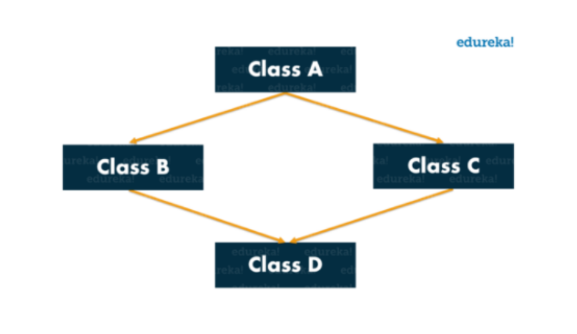
Reasons:

Diamond Problem: One of the main issues with multiple inheritance is the diamond problem, which occurs when a class inherits from two classes that have a common ancestor. It creates ambiguity in the method resolution, as it's unclear which version of a method to call from the common ancestor. This can lead to unexpected behavior and code complexity.

Ambiguity: When the superclasses which the class is inheriting from has the same field/method it becomes unclear which one should be inherited.

HOWEVER, we can achieve multiple inheritance in Java with the help of interfaces.

Diagram for multiple inheritance:



1. Hybrid Inheritance:

Hybrid inheritance is a combination of more than one types of inheritance.

Example:

Just combine multilevel and hierarchical inheritance.

What is super keyword in Java?

Super keyword in java is used to refer to the fields or methods of the immediate superclass of the current class.

Uses:

* To refer to the immediate superclass’ method or field when it is overridden in the child class. When both parent class and child class have fields of the same name we can use the keyword super to access the parent class’ field.
* We can use the super keyword to call the parent class’ constructor.

Code Example1:

class Employee {

    int salary = 10000;

}

public class Programmer extends Employee {

    int salary = 20000;

    void printSalary(){

        System.out.println("Superclass Salary is " + super.salary);

        System.out.println("Subclass salary is " + salary);

    }

    public static void main(String[] args)

    {

        Programmer p = new Programmer();

        p.printSalary();

    }

}

Code Example2:

// Class A with a method named display

class A {

void display() {

System.out.println("Display method in class A");

}

}

// Class B extends class A

class B extends A {

// Method in class B that calls the display method from class A

void callDisplayFromA() {

super.display(); // Using super to call the display method from class A

}

}

public class Main {

public static void main(String[] args) {

// Creating an object of class B

B b = new B();

// Calling the display method from class B

b.display(); // This will call the display method in class B

// Calling the method in class B that calls display from class A

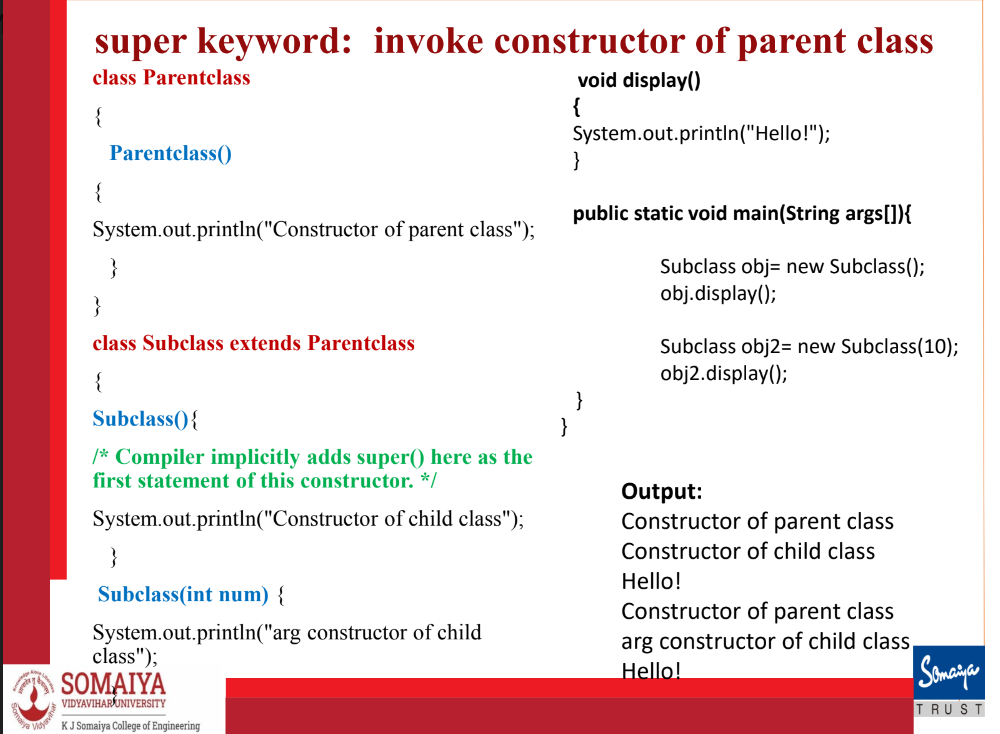
b.callDisplayFromA(); // This will call the display method in class A using super

}

}

Explain use of super to call parent class Constructor.

 Super keyword can also be used to call a parent class constructor. It is very useful in the case where where the parent class has parameterized constructors and you want to perform setup procedures of parent class. Even if you dont add super() in the subclass constructor it is called by default.



You can also use super to call a parameterised constructor.

class Parent {

    int parentValue;

    Parent(int value) {

        this.parentValue = value;

    }

}

class Child extends Parent {

    int childValue;

    Child(int parentValue, int childValue) {

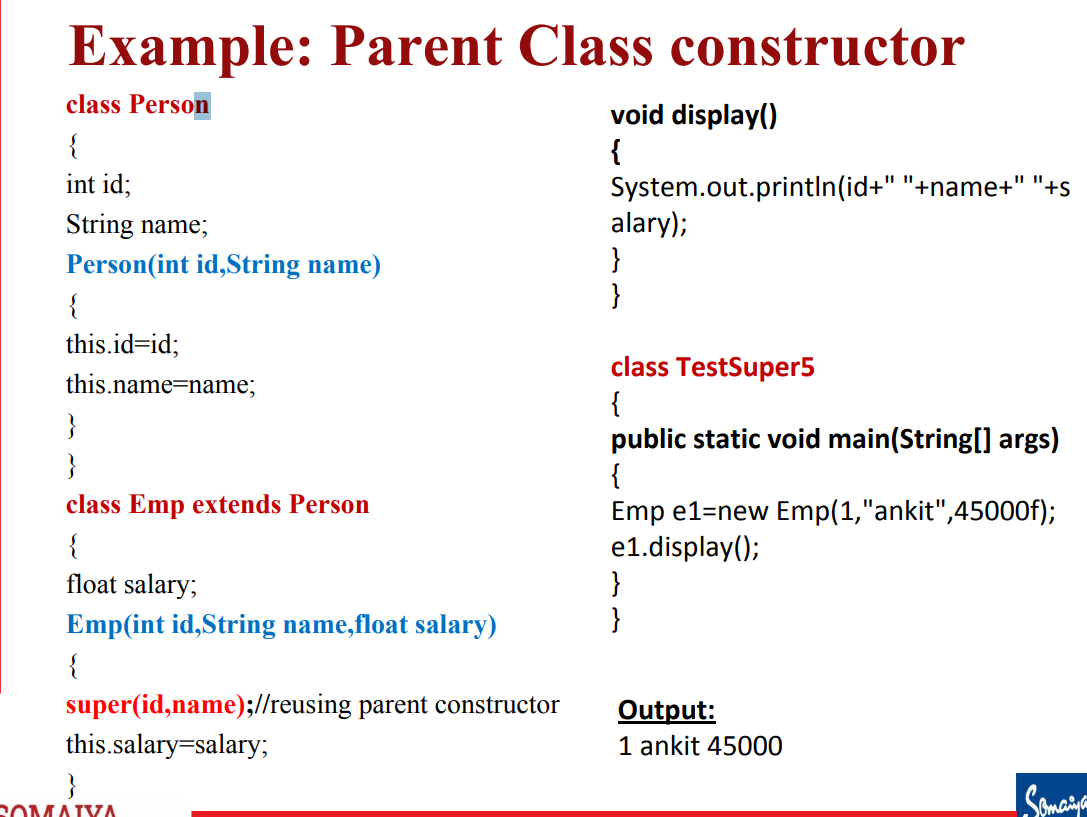
        super(parentValue);

        this.childValue = childValue;

    }

}

Also just like you can make use of superclass field you can call superclass method.



What is method overriding in Java.

Method overriding in java is when you declare a method in the subclass which already exists in the superclass. This is done so that child class can provide its own implementation to a method that was already defined by the superclass.

class Employee {

    int salary = 10000;

    // Overriden Method

    void printSalary()

    {

       System.out.println("Superclass Salary is " + salary);

    }

}

public class Programmer extends Employee {

    int salary = 20000;

    static int bonus = 5000;

    public static void main(String[] args)

    {

        Programmer p = new Programmer();

        p.printSalary(); // Call the overriding method.

        p.superPrintSalary(); // Call the overridden method using super

    }

    // Overriding method

    void printSalary(){

        System.out.println("Subclass salary is " + salary);

    }

    void superPrintSalary(){

        // We can also use super to call overriden method.

        super.printSalary();

    }

}

What is abstraction in java

Abstraction is a process of hiding the implementation details and showing only functionality to the user.  It shows only essential things to the user and hides the internal details. Abstraction allows you to focus on what the object does instead of how it does it.

There are two main ways of achieving abstraction:

Abstract classes(0 to 100%)

Interface(100%)

What is an abstract class and method in java.

Just like you can declare an Object without defining it, you can declare a method without defining it.

Eg. public abstract void draw(int size);

A method that has been declared but not defined is an abstract method.

Any class that has an abstract method is known as an abstract class. An abstract class must be defined with the keyword abstract.

Eg. abstract class MyClass {...}

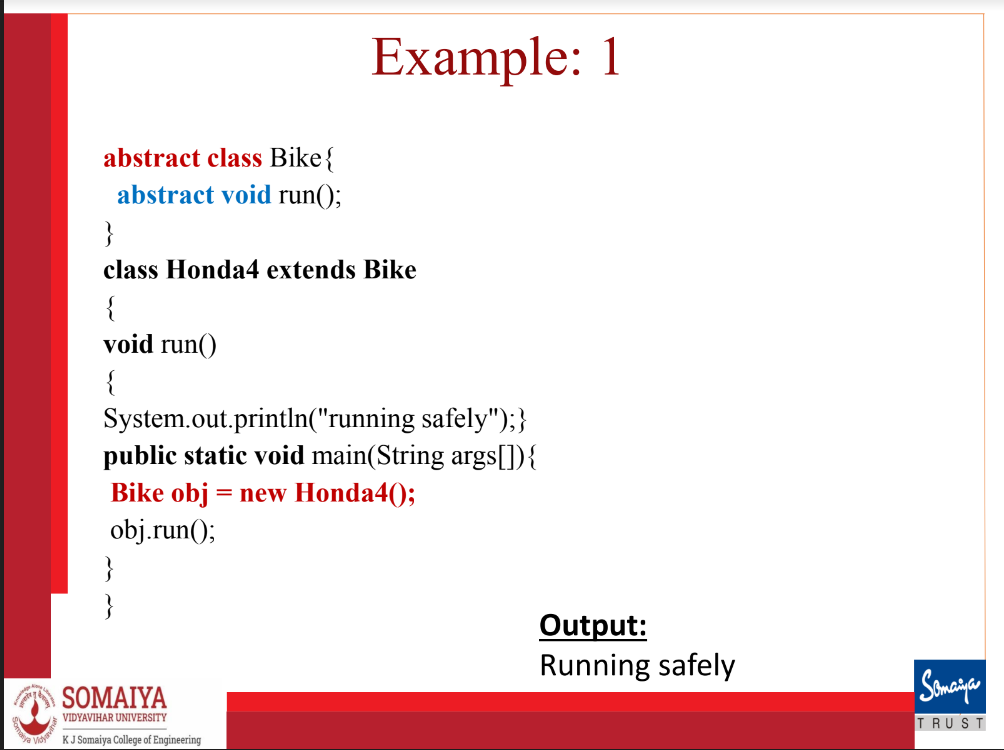
It can have both abstract and non abstract methods

Abstract class can have constructors.

AN ABSTRACT CLASS CANNOT BE INSTANTIATED(You cannot create an object of an abstract class).

You can extend an abstract class. If the extended subclass defines all the abstract methods present in the superclass then it can be instantiated otherwise the subclass must be abstract too.

You can have abstract class without any abstract methods(this is useful to prevent instantiation of that class)



abstract class Employee {

    abstract void displayTitle();

}

class Programmer extends Employee {

    void displayTitle() {

        System.out.println("I am a programmer");

    }

    public static void main(String[] args)

    {

        // Since i have defined all the abstract methods of the superclass

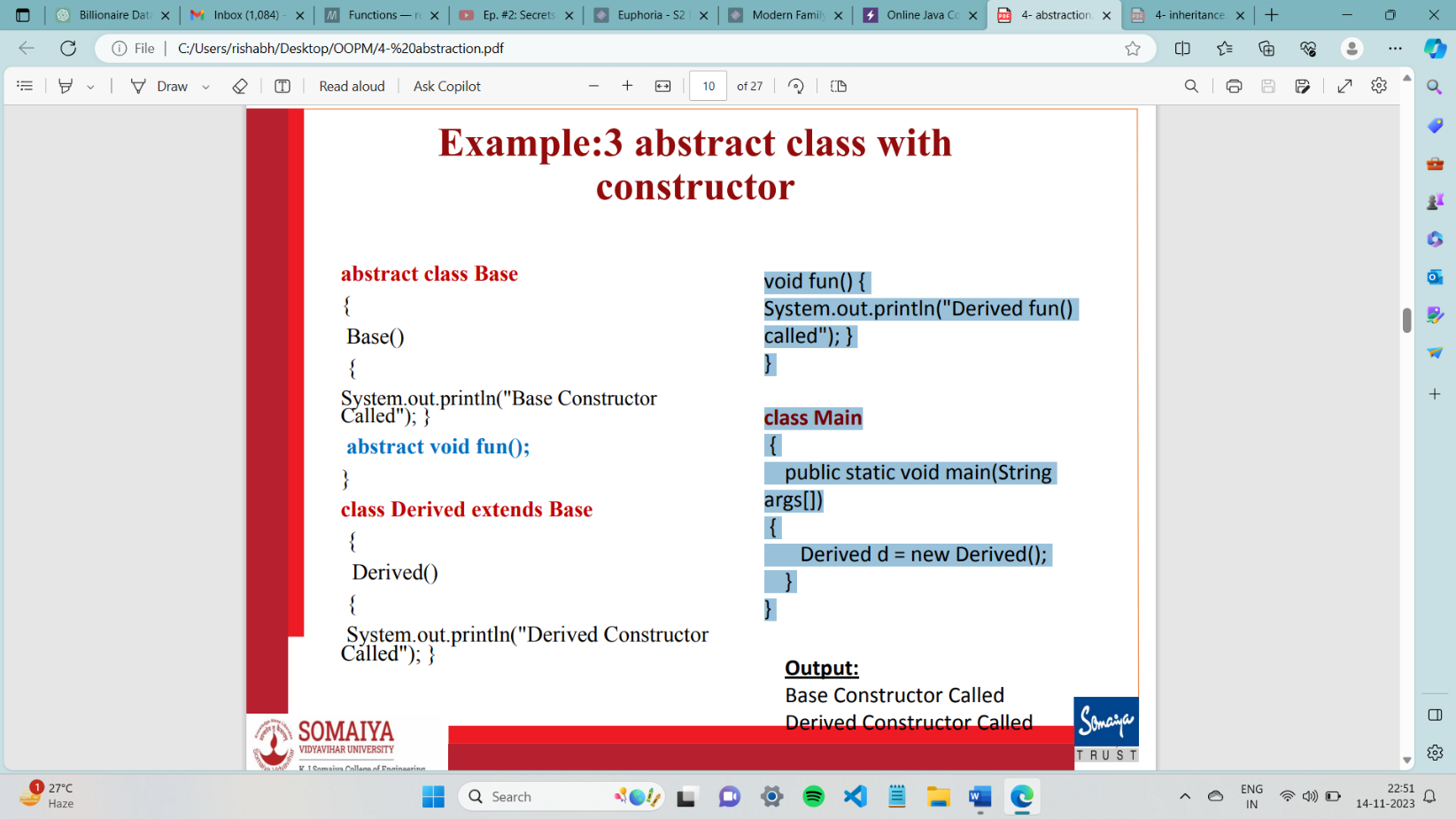
        // the subclass can be instantiated

        Programmer p = new Programmer();

        p.displayTitle();

    }

}



when you create object of subclass, Constructors are called from the top of the inheritance chain (the most superclass) down to the bottom (the most subclass).

What is a class diagram?

A class diagram is a type of diagram that represents the structure and relationships of classes or objects in a system. It identifies the classes and objects, the relationships between them, the attributes of objects and the links and the associations between objects.

A class diagram has boxes which each contain the following 3 sections:

1. Class Name: the topmost section which contains the name of the class
2. Attributes/Fields: The middle section represents the attributes or fields of the objects.
3. Methods: The lowest section contains the methods of the class.

Use of Class Diagrams

• Identify the classes and objects

• Identify relationship among classes and objects

• Identify attributes and operations of objects and links

• Identify associations between objects

• Organize and simplify objects classes using inheritance, generalization and specialization

Different approaches to identifying classes

Noun phase approach (nouns are classes and verbs are methods)

Common class patterns approach

Use case driven, sequence/collaboration approach

Classes responsibilities and collaborators approach

Example box of class diagram and representation of access specifiers:

+ public

# protected

- private

~ package (default)

/ derived

underline static attributes and methods

write <<interface>> on top of interfaces' names

use italics for an abstract class name

Explain the types of relations between classes in a class diagram

Generalisation relation: They represent inheritance relationships, like inheritance between classes and interface implementation. Their hierarchies are drawn top down with arrows pointing from child to parent. If parent is a class, then arrow is solid line with black arrow, if parent is an interface, then it is dashed line and white arrow. IS-A relationship

Association Relation: They represent usage relation between 2 classes. They are of two types:

* Aggregation: Aggregation indicates a relationship where the child can exist separately from their parent class. Example: Automobile (Parent) and Car (Child). So, If you delete the Automobile, the child Car still exist.
* Composition: Composition display relationship where the child will never exist independent of the parent. Example: House (parent) and Room (child). Rooms cannot exist separately without the house.

Write the numbers when not a container otherwise the diamonds

What is an exception and exception handling in Java?

Why do they occur and what is the advantage of exception handling?

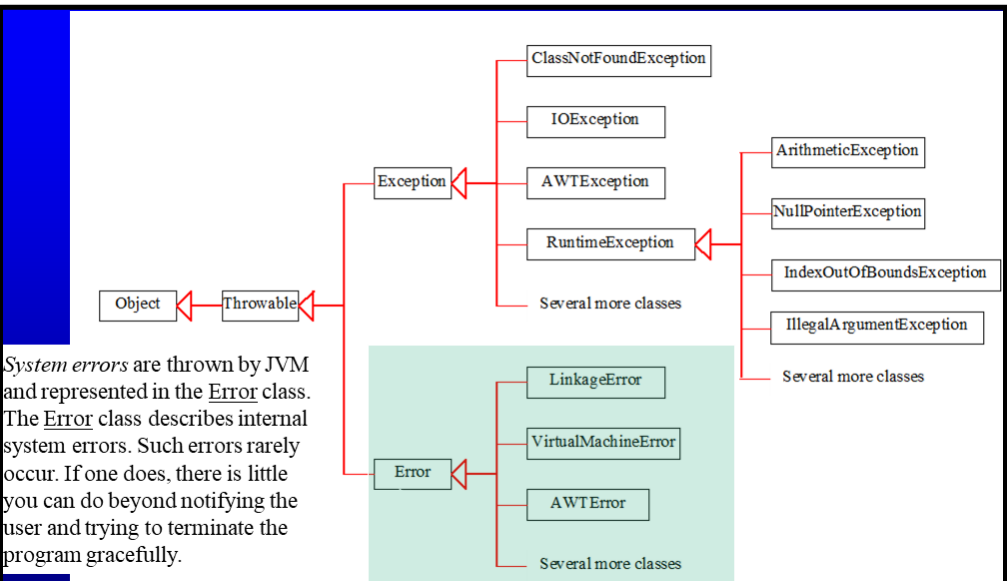
An exception is an unwanted event that occurs which interrupts the normal flow of the program. When an exception occurs the execution of the program gets terminated and we get a system generated error message. In java these exceptions can be handled. Exception handling means providing a meaningful message to the user regarding the error that occurred instead of the system defined message which is hard to understand.

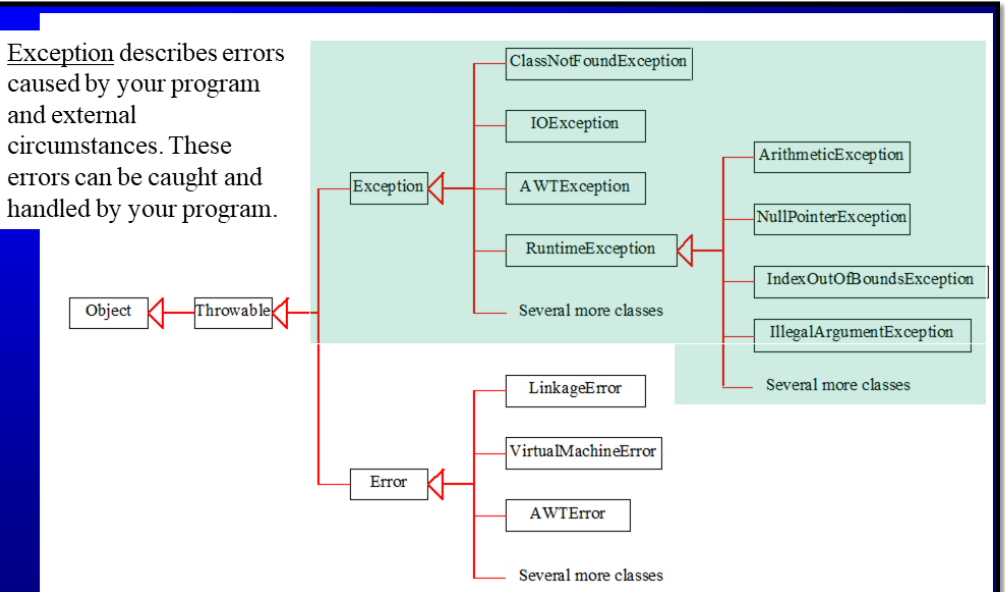
Advantage of exception handling ?

The flow of the program does not break when an exception occurs. Instead of the program terminating we can handle the exception gracefully and allow the remaining lines of code to execute. We can print a customized message for better understanding of user.

Is Exception a subclass or a superclass or both?

The class exception is a subclass of the class Throwable and is a superclass of various exceptions like Runtime exception, ClassNotFoundException and others.





What are the types of exceptions in JAva?

Checked exceptions:

All exceptions other than Runtime Exceptions are known as Checked exceptions as the compiler checks them during compilation to see whether the programmer has handled them or not. If these exceptions are not handled/declared in the program, you will get compilation error Eg. ClassNotFoundException, IOException

Unchecked Exceptions

These exceptions are not checked at compile time so compiler does not check whether they have been handled or not, so it’s the responsibility of the programmer to handle these exceptions and provide a safe exit. Runtime Exceptions and their subclasses are known as Unchecked Exceptions. Eg. ArrayOutOfBoundsException, NullPointerException, ArithmeticException

What are try and catch blocks in Java?

A try block contains the set of statements where an exception could occur. A try block if always followed by one or multiple catch blocks. Try block cannot exist without catch block.

A catch block is where you handle the exceptions that are generated in the try block. It cannot exist alone it must exist with a try block. When an exception occurs in try block, the corresponding catch block that handles that particular exception executes.

SYNTAX: catch (exception(type) e(object))

Example:

public class Main {

    public static void main(String[] args)

    {

        try {

            int x = 50/0;

        }

        catch (ArithmeticException e)

        {

            System.out.println("Exception: " + e);

        }

    }

}

What are some common examples of exceptions occurring in java.

* ArithmeticException : trying to divide by 0
* NullPointerException is thrown when program attempts to use an object reference that has the null value • Invoking a method from a null object. • Accessing or modifying a null object’s field. • Taking the length of null, as if it were an array. • Accessing or modifying the slots of null object, as if it were an array.
* String s=null; System.out.println(s.length());
* NumberFormatException is a runtime exception that occurs when you try to convert a string to a numeric type (like int, double, etc.), but the string does not have the appropriate format for the conversion. Causes- string might be null or empty, there might be leading or trailing space, string might be alphanumeric
* Example: Integer.parseInt(null)
* The ArrayIndexOutOfBoundsException occurs whenever we are trying to access any item of an array at an index which is not present in the array. In other words, the index may be negative or exceed the size of an array

An example of using multiple catch blocks with one try block.

public class Main {

    public static void main(String[] args)

    {

        try {

            int x = 50/0;

        }

        catch (ArithmeticException e)

        {

            System.out.println("Exception: " + e);

        }

        catch (Exception e) //generic exception that catches all exceptions

        {

            System.out.println("Base Exception: " + e);

        }

    }

}

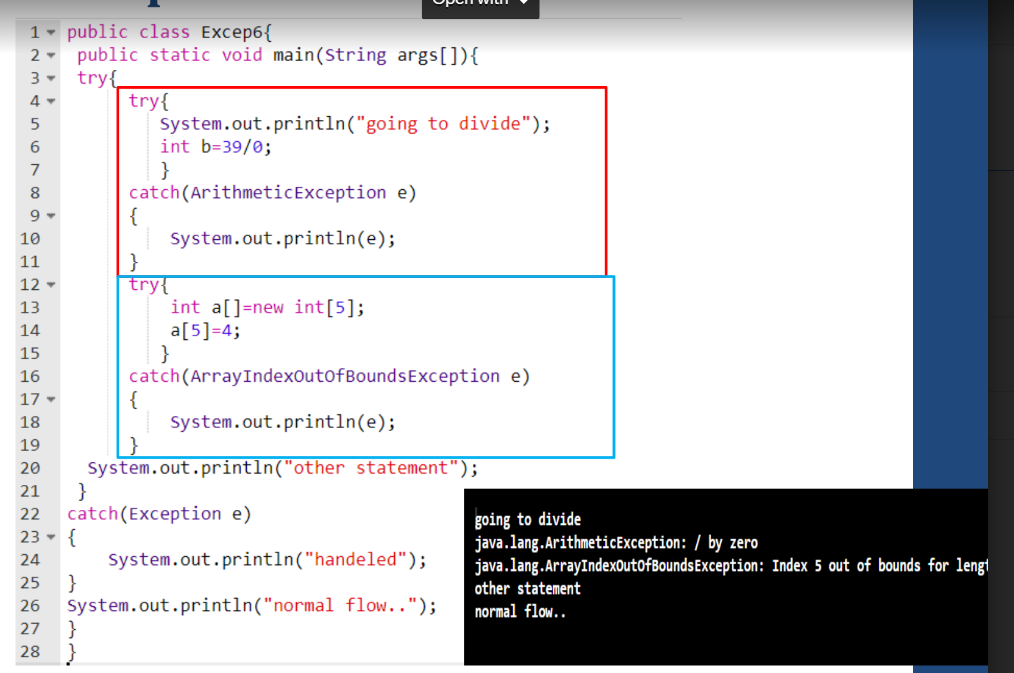
What are nested try catch blocks and can you have a try block without catch block inside another try block?

When a try catch block is present in another try block then it is called the nested try catch block.

• Each time a try block does not have a catch handler for particular exception, then the catch blocks of parent try block are inspected for that exception, if match is found then that catch block executes.

• If neither catch block nor parent catch block handles exception then the system generated message would be shown for the exception, similar to what we see when we don’t handle exception.

The checking moves outwards from child to parent.



What is finally block in Java?

It Contains all the crucial statements that must be executed whether exception occurs or not.

• The statements present in this block will always execute regardless of whether exception occurs in try block or not such as closing a connection, stream , file etc.

try {

//Statements that may cause an exception

}

catch {

//Handling exception

}

finally {

//Statements to be executed

}

A finally block must be associated with a try block, you cannot use finally without a try block. You should place those statements in this block that must be executed always.

Finally block is optional, however if you place a finally block then it will always run after the execution of try block.

In normal case when there is no exception in try block then the finally block is executed after try block. However if an exception occurs then the catch block is executed before finally block.

The statements present in the finally block execute even if the try block contains control transfer statements like return, break or continue.

Finally is executed first and then compiler error is shown incase we are not handling exception. See slides 38 39

What is throw keyword in java?

Throw keyword is used for throwing custom exceptions in already existing Exception classes We can define our own set of conditions or rules and throw an exception explicitly in a method using throw keyword.

• Syntax of throw keyword:

throw new exceptionname("error message");

Example:

public class Main {

    static int div(int a, int b)

    {

        if(b > a)

        {

            throw new ArithmeticException("Divisor is greater than dividend");

        }

        return a/b;

    }

    public static void main(String[] args)

    {

        int x = div(10, 20);

    }

}

What is the throws keyword in java?

the throws keyword is used in the method signature to indicate that a particular method might throw one or more specified exceptions during its execution. When a method is declared with throws, it means that the method is not handling the exceptions itself, and it is leaving the responsibility of handling those exceptions to the caller or the calling method. Throws is useful for code readability in larger codebases to warn that a method might throw an exception.

class Example {

static void myMethod() throws CustomException {

// method body

if (/\* some condition \*/) {

throw new CustomException("An error occurred");

}

}

public static void main(String[] args) {

try {

myMethod();

} catch (CustomException e) {

// Handle the exception

System.out.println("Caught CustomException: " + e.getMessage());

}

}

}

What are user-defined custom exceptions in Java?

In Java, we can create our own exceptions that are derived classes of the Exception class. Creating our own Exception is known as custom exception or user-defined exception.Using the custom exception, we can have your own exception and message.

Following are few of the reasons to use custom exceptions: • To catch and provide specific treatment to a subset of existing Java exceptions. • Business logic exceptions: These are the exceptions related to business logic and workflow

In order to create custom exception, we need to extend Exception class that belongs to java.lang package. we need to passed a string to the constructor of superclass i.e. Exception

Example:

class InvalidDivisionException extends Exception {

    InvalidDivisionException(String s)

    {

        super(s);

    }

}

public class Main {

    static int div(int a, int b) throws InvalidDivisionException

    {

        if(b > a)

        {

            throw new InvalidDivisionException("Divisor is greater than dividend");

        }

        return a/b;

    }

    public static void main(String[] args)

    {

        try {

            int x = div(10, 20);

        }

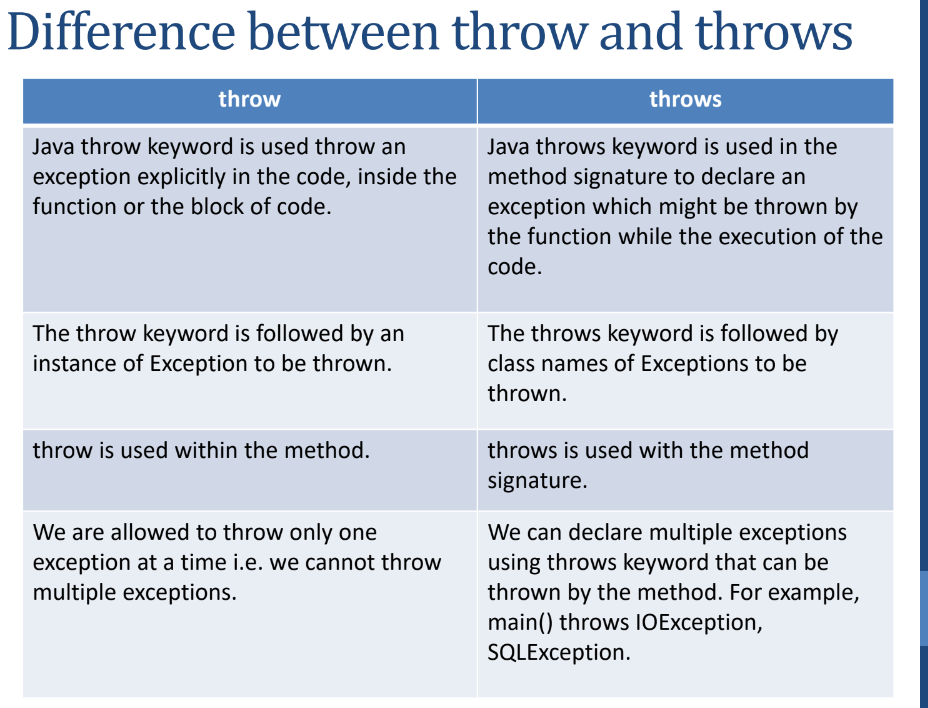
        catch (InvalidDivisionException e) {

            System.out.println("Galat hai bhai " + e);

        }

    }

}



What is a package?

A package can be defined as a grouping of related types, classes, interfaces and enumerations providing access protection and namespace management and achieving data encapsulation

What are the advantages of packages?

**Grouping**: Can be used to categorise the classes and interfaces and make locating similar classes easier.

**Access Protection:** Packages provide a way to control access to classes and other members. Members of a package can have different access modifiers (e.g., public, private, protected). This allows you to control which classes are accessible from outside the package and which are not.

**Namespace Management:** Packages help in managing the namespace by avoiding naming conflicts. Two classes with the same name can coexist if they belong to different packages.

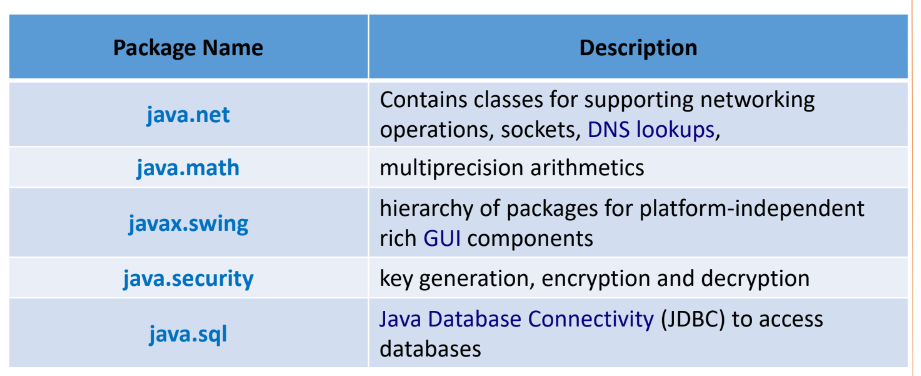
**Data encapsulation and preventing unauthorized access:** While a package may consist of many classes, it's common to expose only a subset of them externally. This means that some classes are designed for internal use within the package and are not meant to be accessed directly by other packages or classes. This is a form of information hiding and helps in encapsulating the implementation details.

What are the categories of packages?

* Built in packages- these are standard packages which come inbuilt as a part of java runtime environment.
* User-defined packages: These are custom packages defined by the user to bundle together related classes and interfaces.

Give some examples of built in packages in Java.





What are user defined packages in Java and how to create one?

A user defined package in java is a custom package defined by the programmer to bundle together related classes and packages. To create a package, have a package statement at the top of the file.

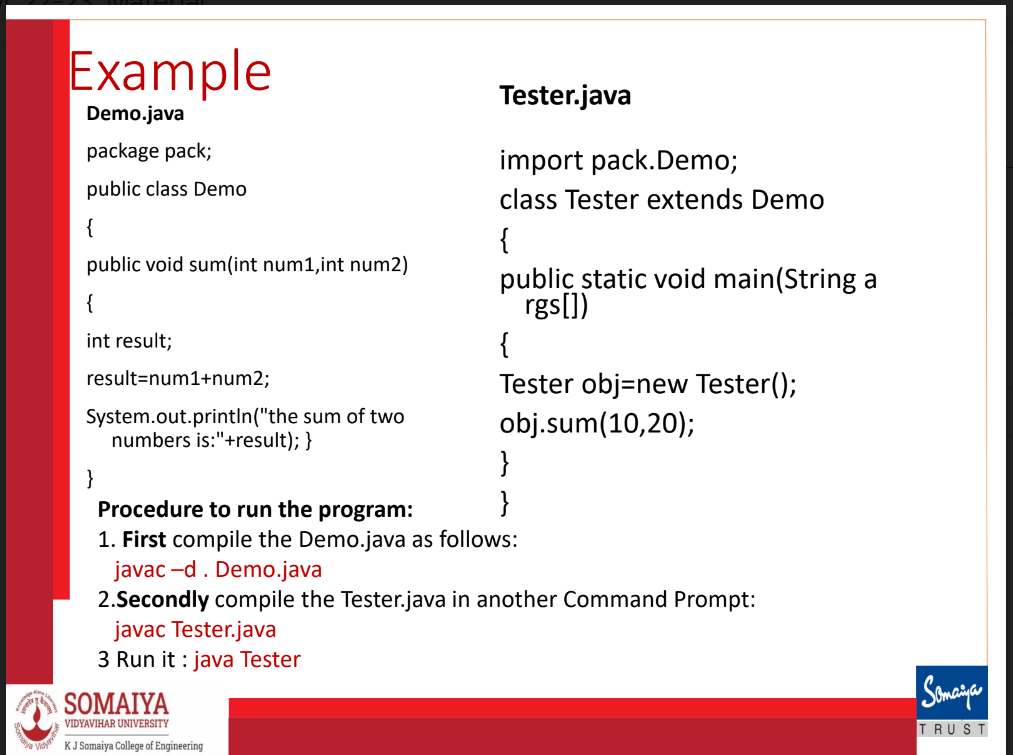
Eg.

package engineering;

public class abc {

// Rest of code

}



What is a subpackage?

Package inside the package is called the subpackage. It should be created to categorize the package further. The packages that come lower in the naming hierarchy are called "subpackage"

• The standard of defining package is package.subpackage

**MODULE 6: Multithreading**

REFER TO MAAMS PPT FOR NOTES

\*find these topics from notes or outside\*

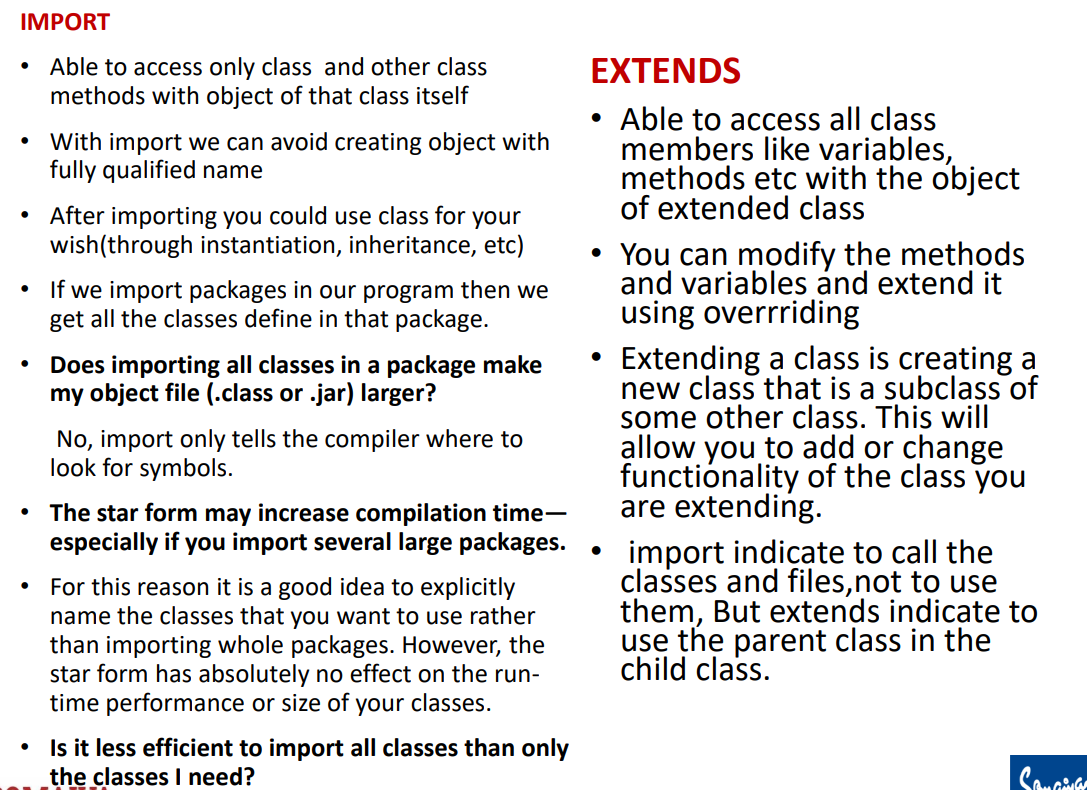
**Thread life cycle**

**Multithreading advantages and issues**

**Simple thread program**

**Thread synchronization.**

**ABSTRACT**



| **Next()** | **NextLine()** |
| --- | --- |
| It read input from the input device till the space character. | It read input from the input device till the line change. |

int day = 3;

switch (day) {

case 1:

case 2:

case 3:

case 4:

case 5:

System.out.println("Weekday");

break;

case 6:

case 7:

System.out.println("Weekend");

break;

default:

System.out.println("Invalid day");

}

The **break** statement in Java is used to terminate the innermost loop that encloses it. If you have nested loops, a **break** statement will only terminate the loop in which it is directly placed, not all the enclosing loops.

public class Main

{

public static void main(String... ar)

{

// loop2:

for(int i=0;i<2;i++)

for(int j=0;j<5;j++)

{

if(j==2)

break;

System.out.println("i = "+i);

System.out.println("j = "+j);

}System.out.println("Out of the loop");

}

}

Output: i = 0

j = 0

i = 0

j = 1

i = 1

j = 0

i = 1

j = 1

Out of the loop

In Java, the **break** with label statement is used to terminate the execution of a labelled statement, allowing you to break out of nested loops or switch statements.

label:

for (int i = 0; i < 5; i++) {

for (int j = 0; j < 5; j++) {

if (some Condition) {

break label; // This will break out of the outer loop labelled as label

}  
}

}

In this example, **label** is a user-defined label that is associated with the outer loop. When the **break label;** statement is encountered, the control flow will exit both the inner and outer loops. In many cases, alternative approaches, such as using Boolean flags or restructuring your code, can lead to more maintainable solutions. Labelled breaks are more commonly used in situations where breaking out of nested structures is necessary.

In Java, the **continue** statement is used to skip the rest of the code inside a loop for the current iteration and move on to the next iteration. It is commonly used in loops (such as **for** and **while** loops) to skip the remaining code within the loop body and proceed to the next iteration based on a certain condition. The **continue** statement is useful in situations where you want to skip certain iterations of a loop based on a specific condition without executing the remaining code for that iteration.

Java does not have explicit pointers like some other languages (e.g., C or C++). Instead, Java relies on references to objects.

Object References:

In Java, when you create an object using the new keyword, you get a reference to that object. This reference is a variable that points to the memory location where the object is stored.

No Direct Pointers:

Unlike languages like C or C++, Java abstracts away the concept of explicit pointers. Developers work with objects through references, and they don't have direct access to memory addresses.

Garbage Collection:

Java has automatic memory management through a process called garbage collection. Objects that are no longer reachable are automatically identified and removed by the garbage collector. This helps prevent memory leaks.

Null References:

Java references can be null, indicating that they don't point to any object. This helps in handling situations where an object is not yet assigned or has been intentionally set to null.

Why References Instead of Pointers:

Safety:

Pointers in languages like C and C++ can be prone to errors such as accessing invalid memory locations, leading to bugs and security vulnerabilities. Java's use of references adds a layer of safety by abstracting memory management.

No Memory Leaks:

Java's automatic garbage collection eliminates the need for manual memory management. Developers don't have to worry about deallocating memory, reducing the likelihood of memory leaks.

Simplified Syntax:

References in Java are used without the need for pointer arithmetic or explicit memory manipulation. This simplifies the language syntax and makes Java code more readable.

Portability:

Java is designed to be platform-independent. Using references instead of pointers allows Java programs to run on any platform with a Java Virtual Machine (JVM) without concerns about low-level memory details.

Encapsulation:

The use of references helps encapsulate the internal details of an object. Developers interact with objects through references, and the underlying memory management is abstracted away.

In Java, the garbage collector (GC) is a part of the Java Virtual Machine (JVM) responsible for automatic memory management. Its primary function is to reclaim memory occupied by objects that are no longer reachable or in use by the program. The garbage collector allows Java developers to avoid manual memory management tasks, such as explicit memory deallocation, reducing the risk of memory leaks and improving overall program robustness.

Here are key aspects of how the garbage collector works in Java:

Automatic Memory Management:

In Java, developers use the new keyword to dynamically allocate memory for objects. The garbage collector automatically identifies and reclaims memory that is no longer referenced by the program.

Reachability:

Objects that are reachable from the root of the object graph are considered live, and their memory is not eligible for garbage collection. The root objects include local variables, static variables, and active threads.

Mark-and-Sweep Algorithm:

The garbage collector typically uses a mark-and-sweep algorithm to identify and reclaim memory. It involves two main phases:

Mark Phase: Traverses the object graph from the roots, marking reachable objects.

Sweep Phase: Identifies and frees memory occupied by unmarked (unreachable) objects.

By automating memory management, the garbage collector in Java simplifies the programming process, reduces the risk of memory-related errors, and contributes to the overall reliability of Java applications.

JVM is the runtime environment that executes Java bytecode.

JRE includes the JVM and libraries, providing a runtime environment for Java applications.

JDK includes the JRE plus development tools, making it a complete software development kit for building Java applications.

To run Java applications, you need the JRE. To develop Java applications, you need the JDK. The JDK includes the JRE, so installing the JDK also provides the runtime environment necessary for running Java applications.

Java Source Code:

Developers write Java source code using a text editor or an integrated development environment (IDE). Java source code files typically have a .java extension.

Compilation:

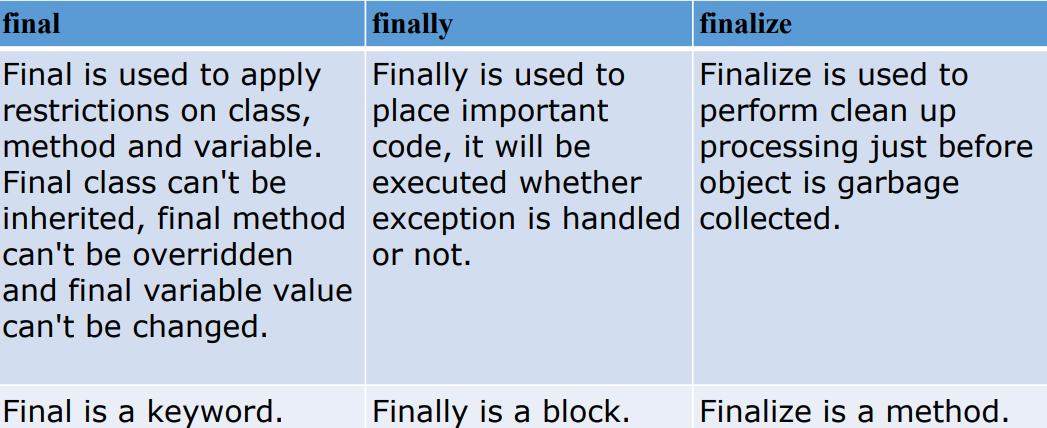
The javac compiler is used to compile Java source code into bytecode. It translates the human-readable Java source code into a platform-independent intermediate representation known as Java bytecode. The bytecode is stored in files with a .class extension.The produced bytecode is not native machine code but is instead a set of instructions for the Java Virtual Machine (JVM). This bytecode can be executed on any system with a compatible JVM.

Java Virtual Machine (JVM):

The JVM interprets or compiles the bytecode into native machine code at runtime. This step ensures that Java applications are platform-independent. The JVM is responsible for executing Java bytecode and providing a runtime environment for Java applications.

The JRE includes the JVM along with libraries and other runtime components necessary for executing Java applications. When running a Java application, the JRE is needed to provide the necessary runtime support. The javac compiler is part of the JDK, which is a comprehensive software development kit. Developers use the JDK for both compiling Java source code and developing Java applications. The JDK includes the JRE, development tools, and additional libraries.

In summary, the javac compiler is responsible for translating Java source code into platform-independent bytecode. This bytecode can then be executed by the Java Virtual Machine, making Java applications versatile and capable of running on various platforms without modification. The JDK, including the javac compiler, is a complete development kit for creating and running Java applications.



In Java class diagrams, both generalization and association are important concepts that help represent the relationships between classes.

1. **Generalization Relationship:**
   * **Explanation:** Generalization represents an "is-a" relationship between classes. It is used to model inheritance, where one class is a more general version (superclass or parent) and another class is a more specific version (subclass or child). The subclass inherits attributes and behaviors from the superclass.
   * **Example:**

class Animal { // Attributes and methods common to all animals } class Dog extends Animal { // Additional attributes and methods specific to dogs }

1. **Association Relationship:**
   * **Explanation:** Association represents a "has-a" relationship between classes. It signifies that one class is related to another class, usually through the use of fields (attributes) or methods. Associations can be one-to-one, one-to-many, or many-to-many.
   * **Example:**

class Person { String name; Address address; // Association: Person has an Address }

Both generalization and association relationships are crucial for designing and understanding the structure of a software system. They help in creating a clear and concise representation of the relationships between classes, facilitating better communication among developers and ensuring a more maintainable and scalable codebase.

In object-oriented programming, associations between classes can further be classified into two types: Aggregation and Composition. Both represent a "has-a" relationship but differ in terms of the strength of the relationship and the lifecycle of the associated objects.

1. **Aggregation:**
   * **Explanation:** Aggregation is a weak form of association, where one class is associated with another class, but the associated class can exist independently of the source class. It represents a "whole-part" relationship, where the part (associated class) can exist independently of the whole (source class). Aggregation is often represented by a diamond-headed line.
   * **Example:**

class Department { String name; List<Employee> employees; // Aggregation: Department has a list of Employees }

* + **Key Characteristics:**
    - Weak relationship.
    - The associated class can exist independently.
    - Represented by a diamond-headed line in UML diagrams.

1. **Composition:**
   * **Explanation:** Composition is a stronger form of association, where the associated class is part of the source class, and it cannot exist independently. It represents a strong "whole-part" relationship, where the part (associated class) is a vital component of the whole (source class). Composition is often represented by a solid diamond-headed line.
   * **Example:**

class Car { Engine engine; // Composition: Car has an Engine }

* + **Key Characteristics:**
    - Strong relationship.
    - The associated class is a vital part of the source class.
    - Represented by a solid diamond-headed line in UML diagrams.

Aggregation Example:

import java.util.List;

import java.util.ArrayList;

class Department {

String name;

List<Employee> employees; // Aggregation: Department has a list of Employees

public Department(String name) {

this.name = name;

this.employees = new ArrayList<>();

}

public void addEmployee(Employee employee) {

employees.add(employee);

}

}

class Employee {

String name;

public Employee(String name) {

this.name = name;

}

}

public class AggregationExample {

public static void main(String[] args) {

Employee emp1 = new Employee("John Doe");

Employee emp2 = new Employee("Jane Smith");

Department department = new Department("IT Department");

department.addEmployee(emp1);

department.addEmployee(emp2);

// The Department has a list of Employees, but the Employees can exist independently

}

}

Composition Example:

class Engine {

// Engine details

}

class Car {

Engine engine; // Composition: Car has an Engine

public Car() {

this.engine = new Engine();

}

}

public class CompositionExample {

public static void main(String[] args) {

Car myCar = new Car();

// The Car has an Engine, and the Engine is an integral part of the Car

}

}

In the aggregation example, the Department class has a list of Employee objects, but each Employee can exist independently. In the composition example, the Car class has an Engine, and the Engine is created within the Car constructor, indicating a strong composition relationship where the Engine is a vital part of the Car.

Arrow is always in father side whether generalization or composition

Also while redind it you read you read it top to bottom example it as 1 parent contrain 1 or more children. The multiplicity also u read in order top to bottom.