Java and Microservices

Java is platform independent & object oriented language.

Java applications are compiled to byte code and these bytes codes are run by JVM (Java Virtual Machine)

When you install Java you get 2 things

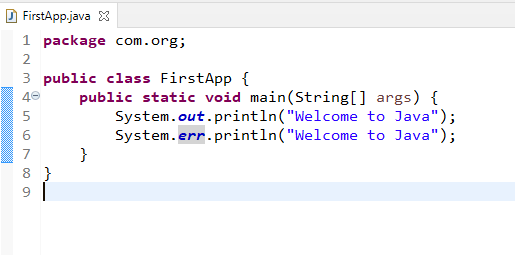
* JDK - Compiler
* JRE - Java Runtime Environment, JVM is part of JRE

Class: Blueprint of an object

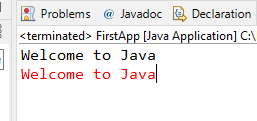
Object: It is an instance of the class

Package: It is a folder that will have all the classes & sub packages, package names follow reverse domain convention i.e., com.org, com.google, com.ibm,

FirstApp.java



Output:



Fundamentals of Java

1. Datatypes
2. Keywords
3. Variables
4. Operators
5. Loops
6. Conditional Statements
7. Classes & Objects

Datatypes: They mention what kind of value a variables stores, there are two types

1. Primitive Datatypes - byte, short, int, long, float, double, char, boolean
2. Non-Primitive Datatypes - Classes, Interfaces, Enums, Arrays

|  |  |
| --- | --- |
| Datatypes | Size in byte |
| byte | 1 |
| short | 2 |
| int | 4 |
| long | 8 |
| float | 4 |
| double | 8 |
| char | 2 |
| boolean | 1 |

By default all the whole numbers are treated as int, but if you want to represent higher precession types you have some suffix

125: It is treated as int

125L: It is tread as long, you can also use lower case ‘l’

20.05: It is treated as double by default

20.05F: It is treated as float, you can also use lower case ‘f’.

byte b1 = 125; // 125 is byte only

short s2 = 350; // 350 is short only

int s3 = 350; // 350 is int

long s4 = 450; // 450 is treated as int

float f = 45.0; // it gives error, because 45.0 is treated as double

float f2 = 45.0f; // 45.0 is treated as float

Keywords: These are reserved words we must not use to create identifiers in that name, some of the keywords are:

int, long, float, short, byte, char, boolean, static, void, public, private, protected, package, class, interface, enum, for, do, while, if, else, return, abstract, default

Operators:

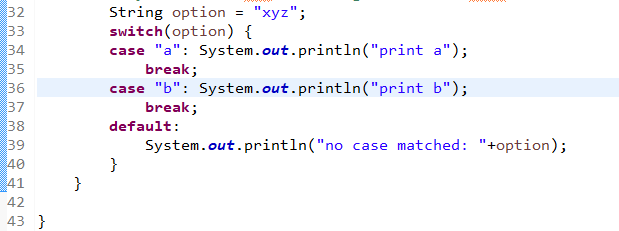
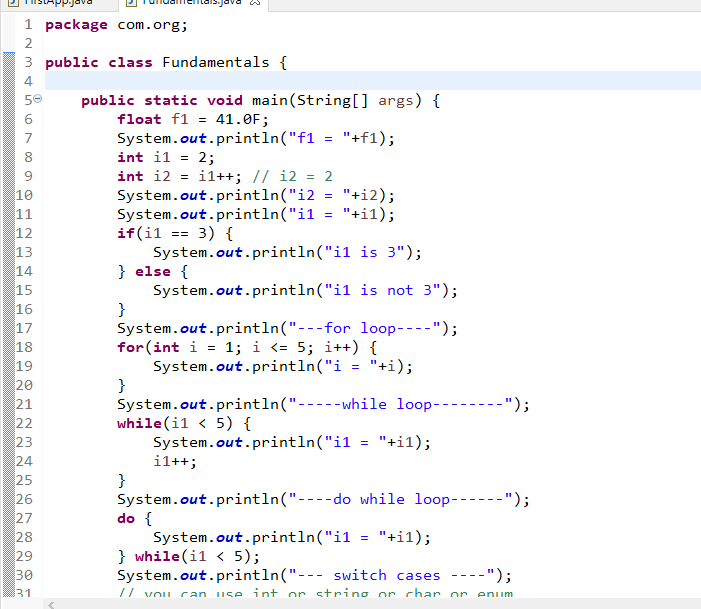
=, ==, !=, <=, >=, ++, --, \*, /, %, &&, ||

Loops:

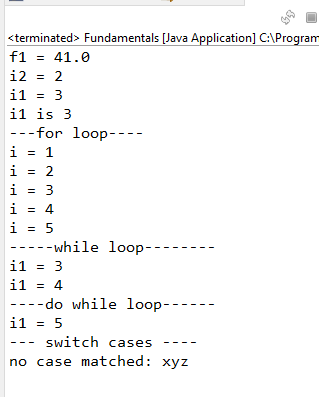
* for
* while
* do while

Conditional statements:

* if
* if else
* if else if else if … else
* switch

Using Fundamentals  


Output:



classes & objects

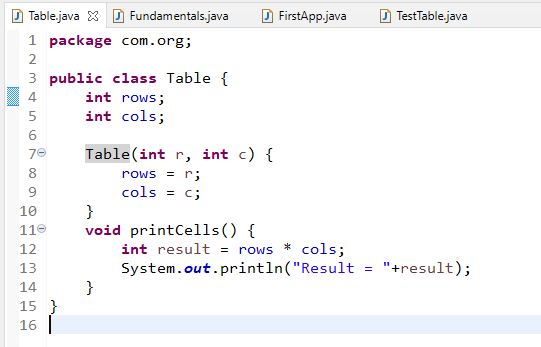
Classes: These represents the structure of the object, it is a blueprint of the object

Object: These are created from the classes, it is an instance of the class

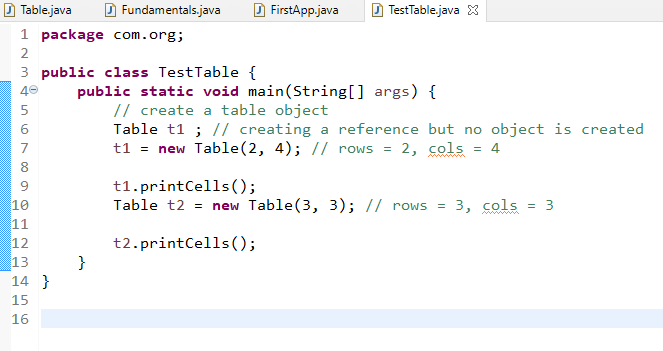
What all the things you can have in a class

1. variables: These store the data
2. methods: These can be any name, they perform actions
3. constructors: same as the class name but used to create the object, initializes the object properties/variables

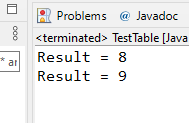
Table.java



TestTable.java



Output:



Note: Java by default creates a constructor i.e., default constructor only in a case where you don’t have any constructors in the class.

Apart from the variables, constructors, methods we can have

static variables & static methods

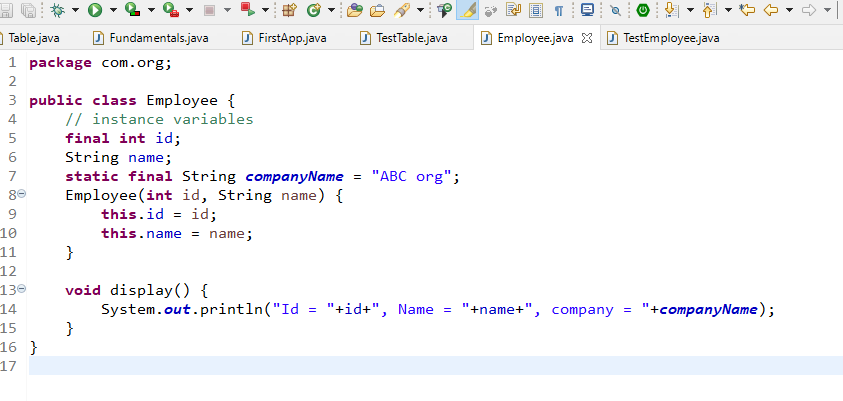
Static: It is part of the class not a part of an object, you can access static members without creating object, you can use class names directly.

Static means common to all the objects, if you modify it is reflected to all the objects, but it can be changed.

How to create constants in Java

final keyword: It can be used to create constants

Employee.java



Final instance variable can be initialized at the time declaration or in constructor,

Static Final variable need to be initialized at the time declaration only,

Final Instance variable is a constant per object

Final static variables are constants for all the objects

OOPs principles

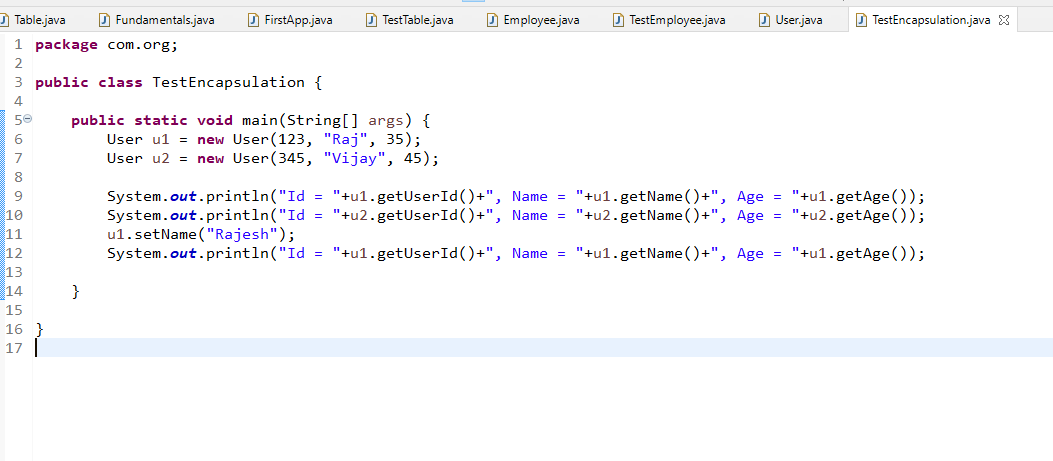
1. Encapsulation
2. Inheritance
3. Polymorphism
4. Abstraction

Encapsulation: Data hiding, where you make variables private and only way can access it is only through the public methods of the enclosing class.

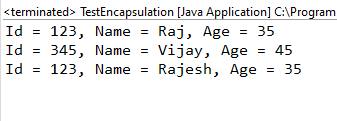
User.java



TestEncapsulation.java



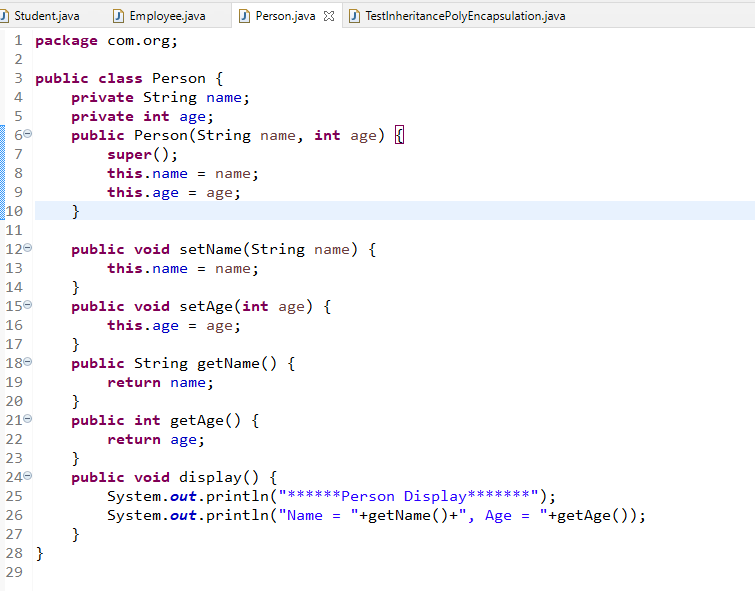
Output:



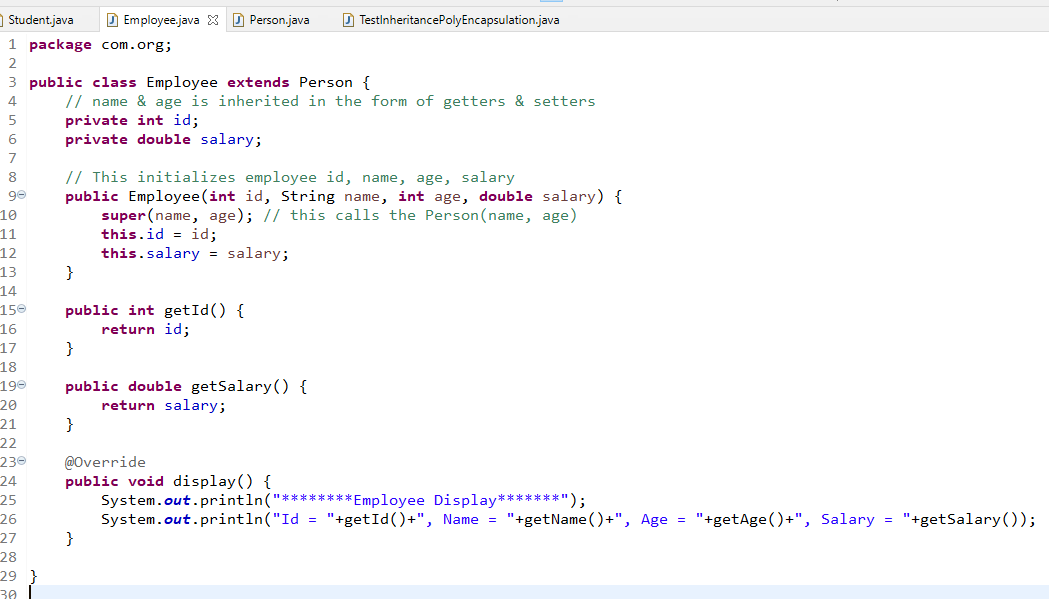
Inheritance: Process of acquiring properties & behaviours from one class to another

Polymorphism: A method with many forms

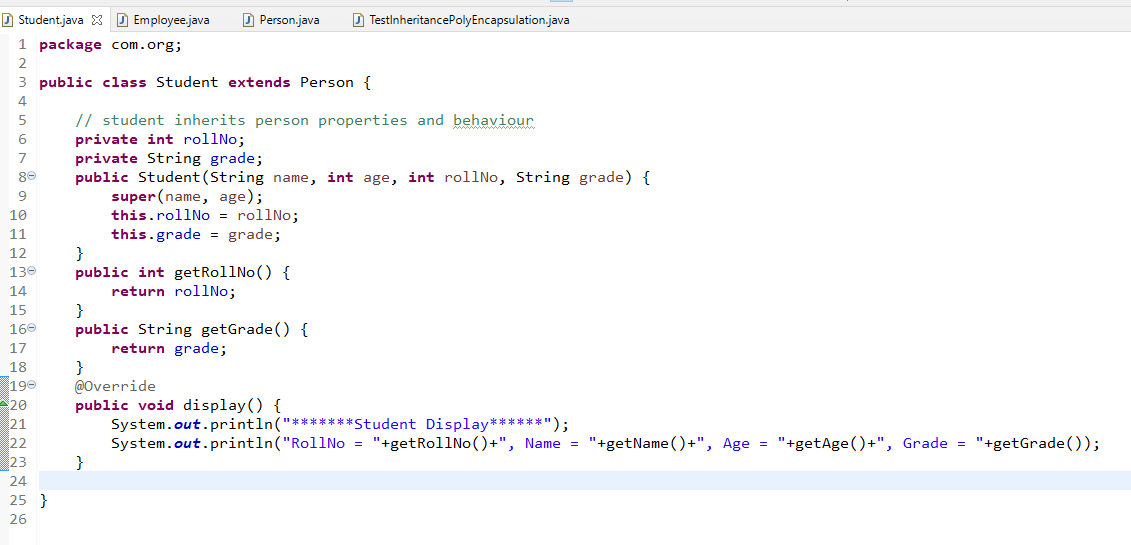
Person.java



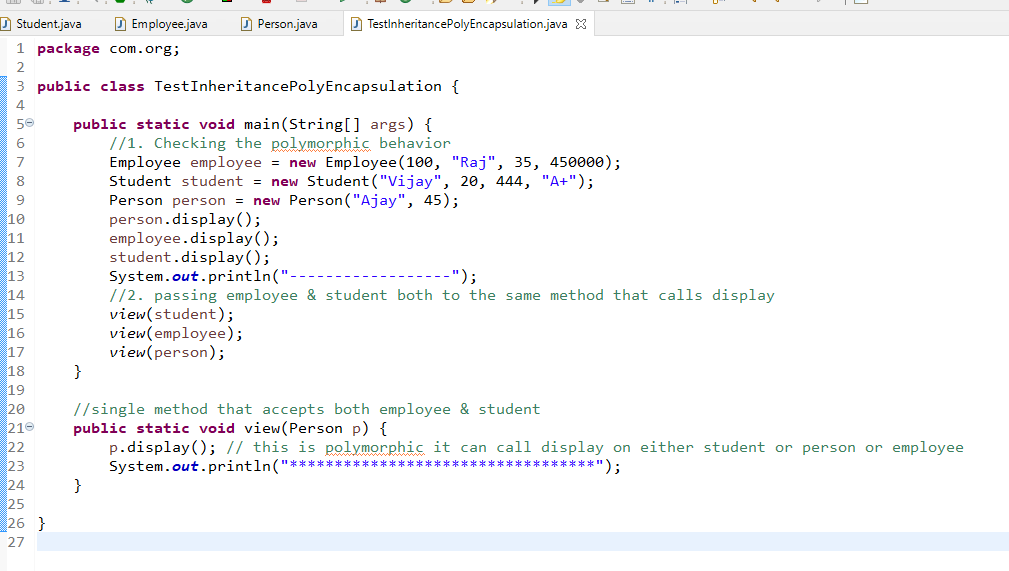
Employee.java



Student.java

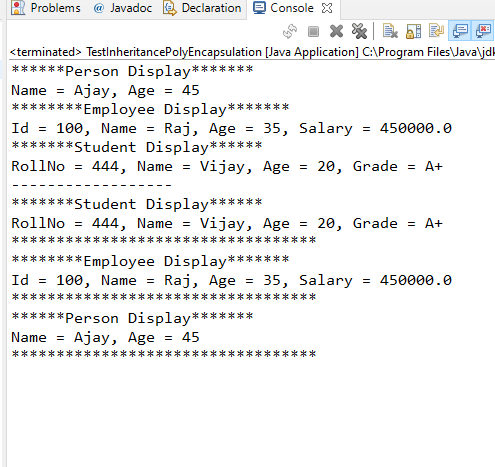


TestInheritancePolyEncapsulation.java



Note: Since Person is an higher type it can accept all of its lower types object, as they are inherited from Person

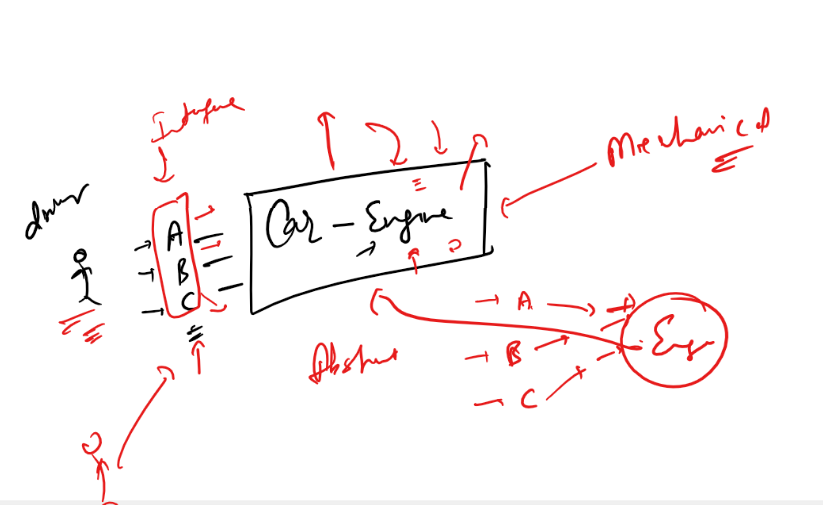
Output:



Activity:

Using interface Person create getName(), getAge() methods and implement it in Student & Employee

Abstraction: Hiding the complexity and showing the necessary details to the User.



Here driver & mechanic uses same object, if mechanic does any change in the engine the driver doesn’t need to change his actions, he can use this object how he was using earlier after modification, even if new driver comes he also knows how to use these interfaces like A, B, C.

Here the engine is complex part that is hidden from the driver, so only the person who has knowledge can repair, driver can just use the interfaces

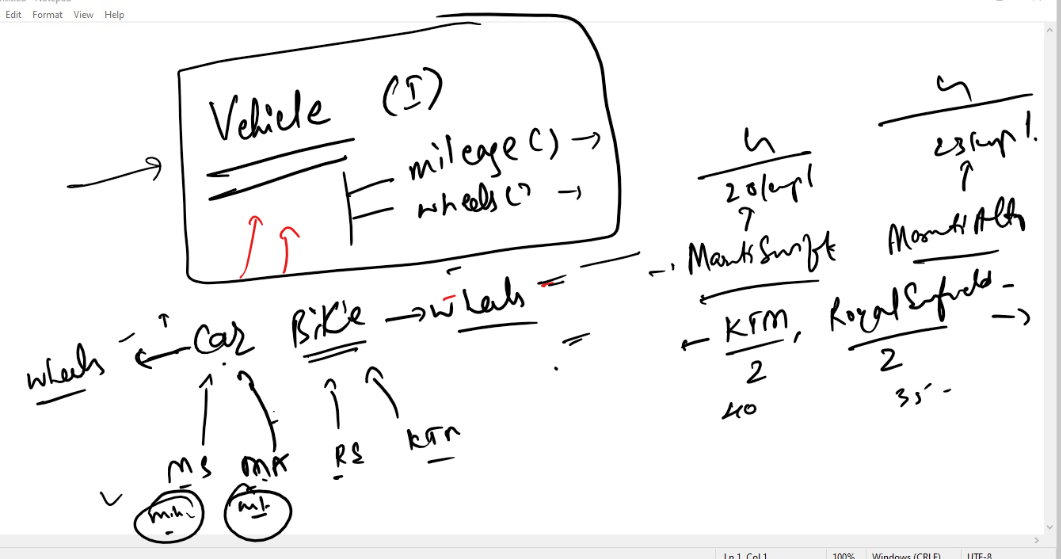
To achieve abstraction we have interfaces

Interface: It is a type of class which will have only abstract methods or incomplete methods

when a class implements interface it must implement all the incomplete methods else class should be declared abstract (which means class is incomplete)

There are 2 ways you can achieve abstraction

1. interface: You will have all the methods abstract
2. abstract class; You will have few abstract & few non-abstract methods



Activity:

Create a vehicle interface with 2 abstract methods mileage() & wheels(),

Create 2 classes Bike & Car and implement Vehicle interface with wheels() method but mileage() must not be implemented.

Create 4 classes like KTM, RoyalEnfield, Swift, Alto, where KTM & RoaylEnfield extends Bike & Swift & Alto extends Car

Create all 4 objects of KTM, RoyalEnfield, Alto & Swift and pass to a single method that takes Vehicle as the reference, from that reference call mileage() & wheels.

Object class:

It is the top most super class for all the classes in Java, by default all the classes extends Object class, if a class doesn’t extend Object then automatically compiler adds extends Object to that class.

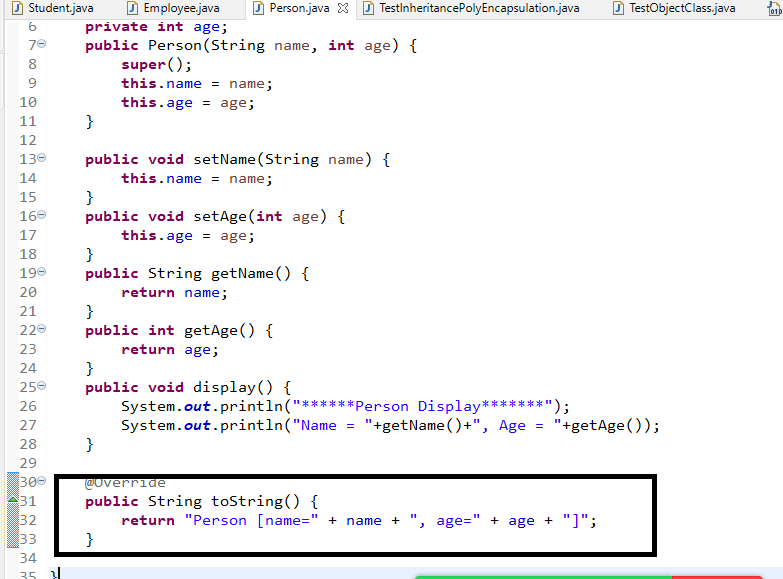
If class A { } is written then compiler treats this as class A extends Object { }

There are few important methods in Object that you can override

1. toString()
2. equals()
3. hashCode()

toString(): It is called automatically when you print object, it returns string representation of the Object.

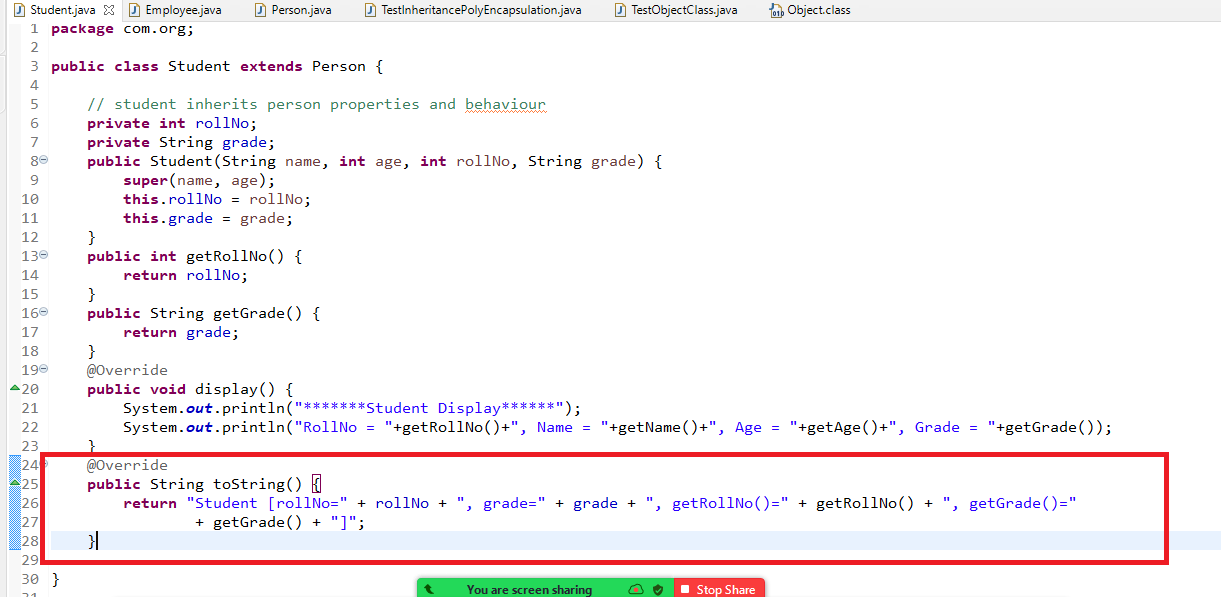
Person.java



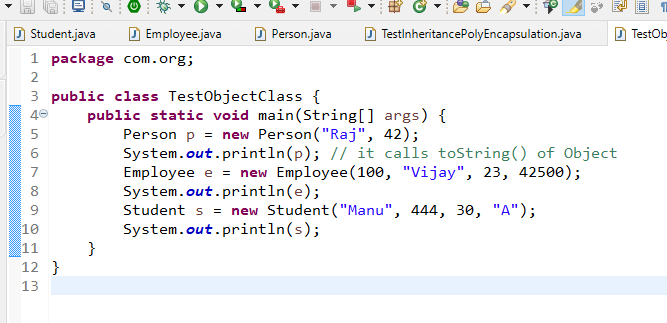
Employee.java



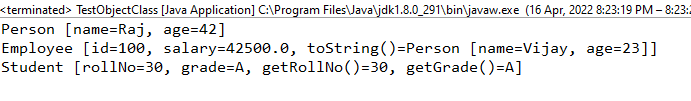
Student.java



TestObjectClass.java



Output:



Predefined classes

<https://docs.oracle.com/javase/8/docs/api/>

<https://docs.oracle.com/javase/tutorial/>

Below are the predefined classes

String, Math, Integer, Double, Exception, RuntimeException, ArithmeticException, System

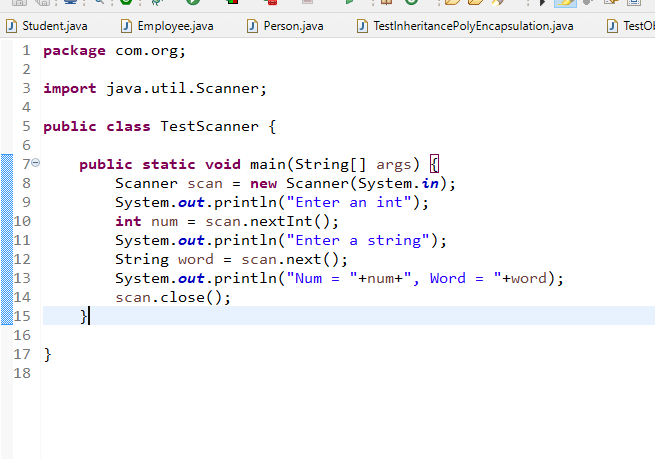
All the above classes you can use without importing, because they are part java.lang, its by default imported

LocalDate, LocalDateTime, LocalTime, Collection, Set, HashSet, LinkedHashSet, TreeSet, Map, TreeMap, LinkedHashMap, List, ArrayList, Vector, Scanner,

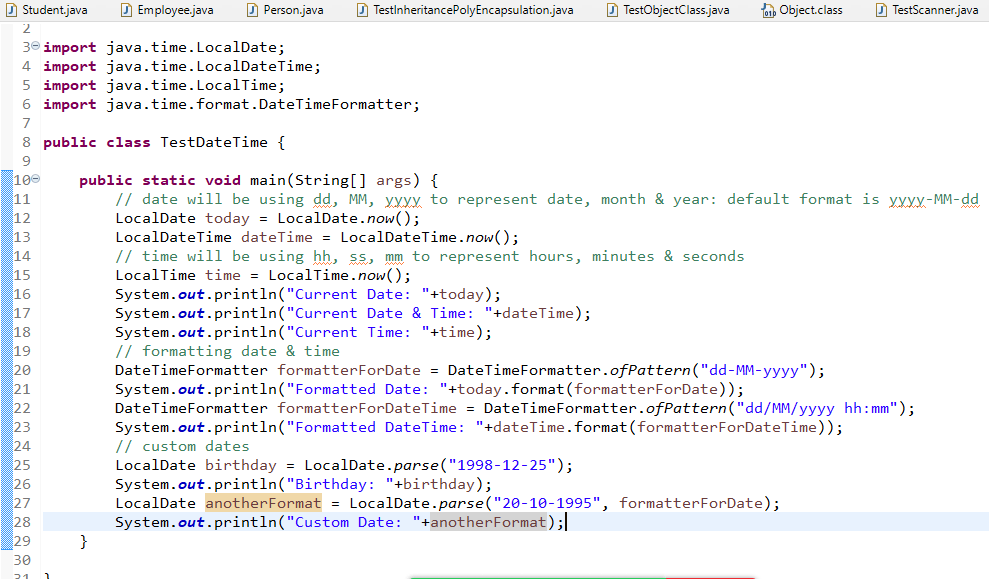
All the above classes are part of different packages, you need to import it

suppose: You want to use Scanner, then it must be imported and its in java.util package

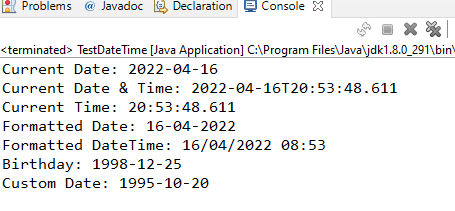
i.e., java.util.Scanner



LocalDate, LocalTime, LocalDateTime: They allow you to work with date & time

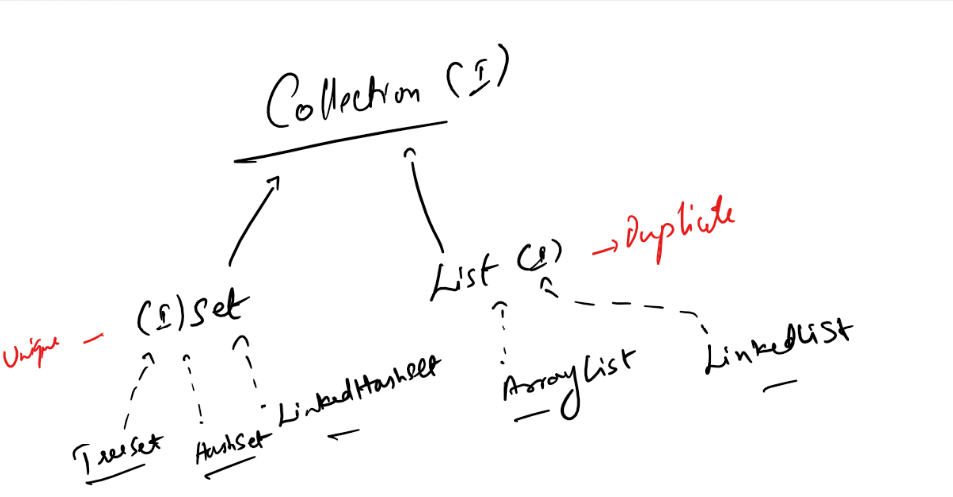


Output:



Collection Framework

It is used to maintain objects in various forms, the Collection Framework



Here Collection is an interface & it has 2 sub interfaces

1. Set: It supports only unique elements
2. List: It supports duplicates & index based

Set has 3 implementations

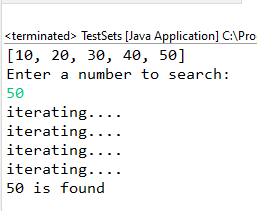
1. TreeSet: It maintains elements Sorted order
2. HashSet: It maintains elements Random Order, but its retrieval faster
3. LinkedHashSet: It maintains elements in Insertion Order

List has 2 implementations

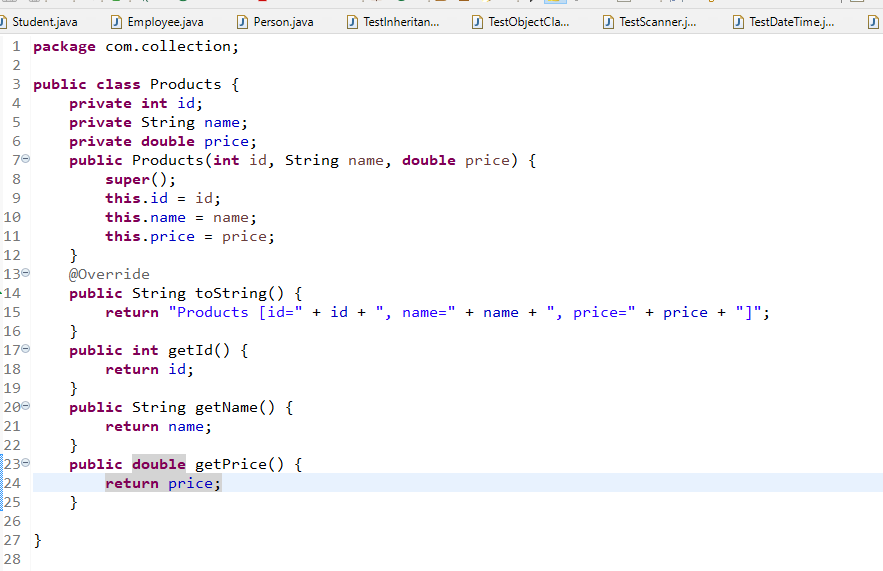
1. ArrayList: It maintains in contiguous memory address
2. LinkedList: It maintains in non-contiguous memory address



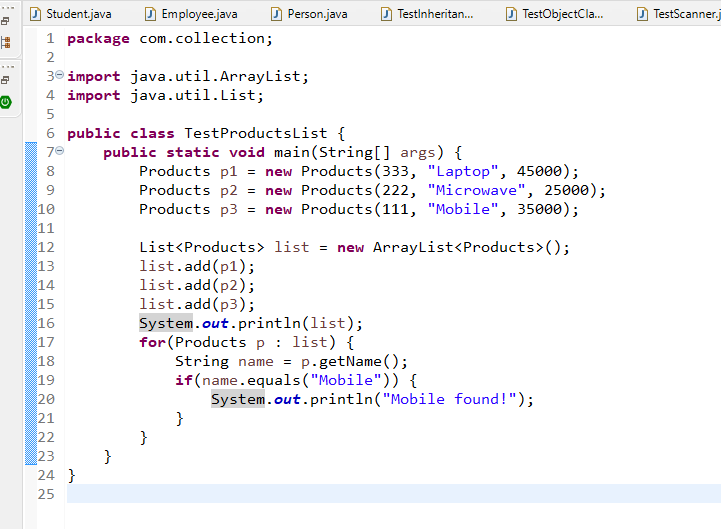
Output:



Products.java

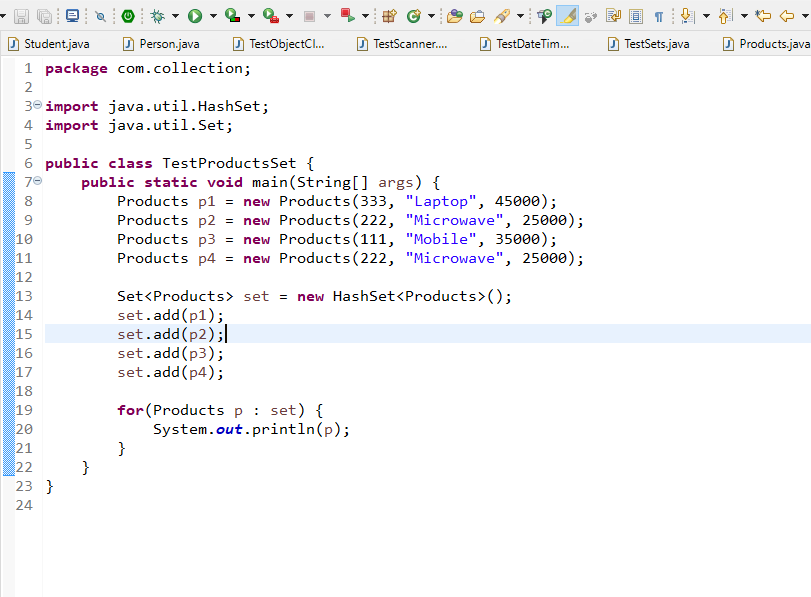


TestProductsList.java

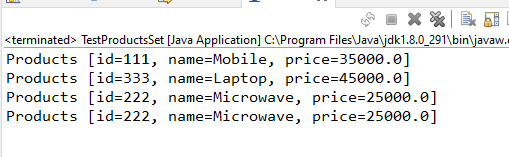


Note: In List you can have duplicate product id, however in case of Set either TreeSet, LinkedHashSet or HashSet they must not support duplicate

TestProductsSet.java



Output:



Note: This Set must not store duplicates, but Set fails to identify on which property it has to identify the duplicates as its custom object, for this you need to override equals() & hashCode() methods of Object in the Products class

Activity: Try the above exercise

Set: It keeps only unique elements, it internally calls two methods of every object

1. hashCode: Helps in identifying bucket number
2. equals: Help to compare

Both equals() & hashCode() are present in Object class, but many classes has overridden equals() & hashCode() for their requirement

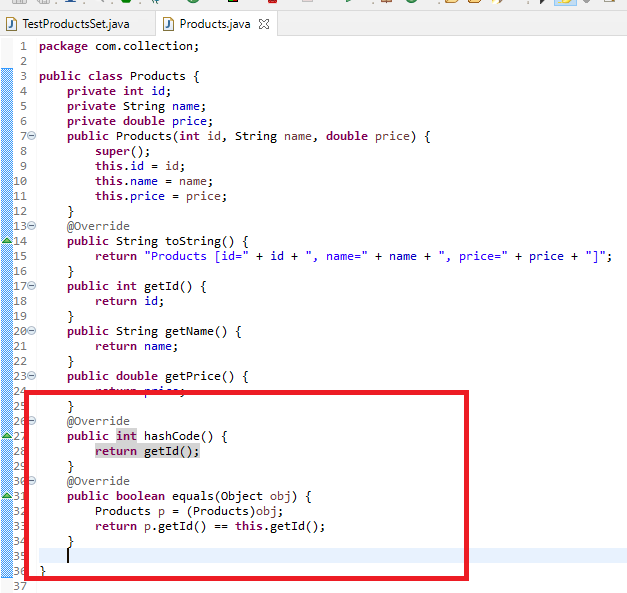
ex: In String equals() & hashCode() is implemented for contents

ex: In Object equals() & hashCode() is implemented for memory address

hashCode() in Object returns memory address in int format

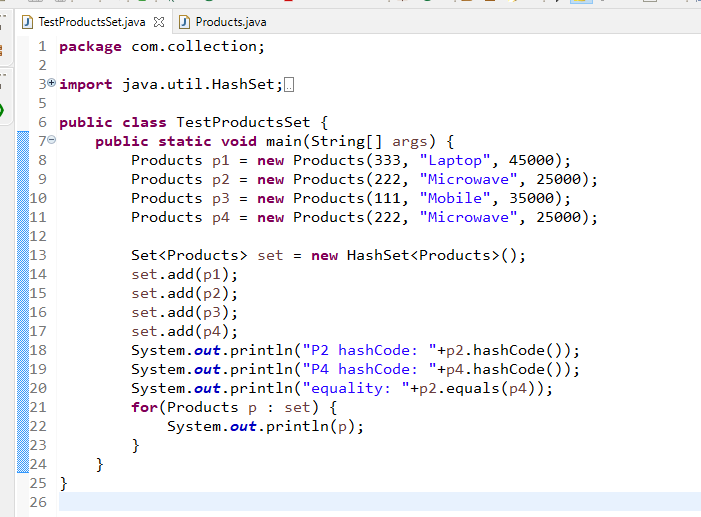
equals() in Object compares 2 objects memory address.

Products.java

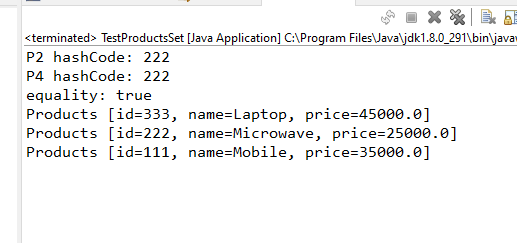


Here equals() & hashCode() works based on product id, so set can call hashCode() and equals() of Products

TestProductsSet.java



Output:

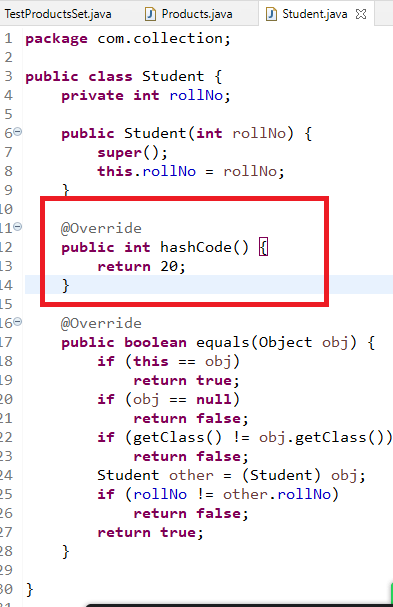


Note: Set creates an Hashtable with some buckets, while adding elements it calls hashCode() to identify the bucket number, if there’s already an element existing in the same bucket then it calls equals() to compare, if equals() returns true, it means element already exists, else doesn’t exist, so it keeps the new element in a linked-list, if the element goes to new bucket then it doesn’t call equals() as there wouldn’t be any element to compare.

What happens if all the elements of Set goes to same bucket?

Performance will degrade, because every element goes to same bucket and every new element is compared with all the elements in the bucket, ex: if the bucket has 1000 elements and a new element is compared with all the 1000 elements.

This happens when hashCode is wrongly implemented, i.e., below way



Exception Handling

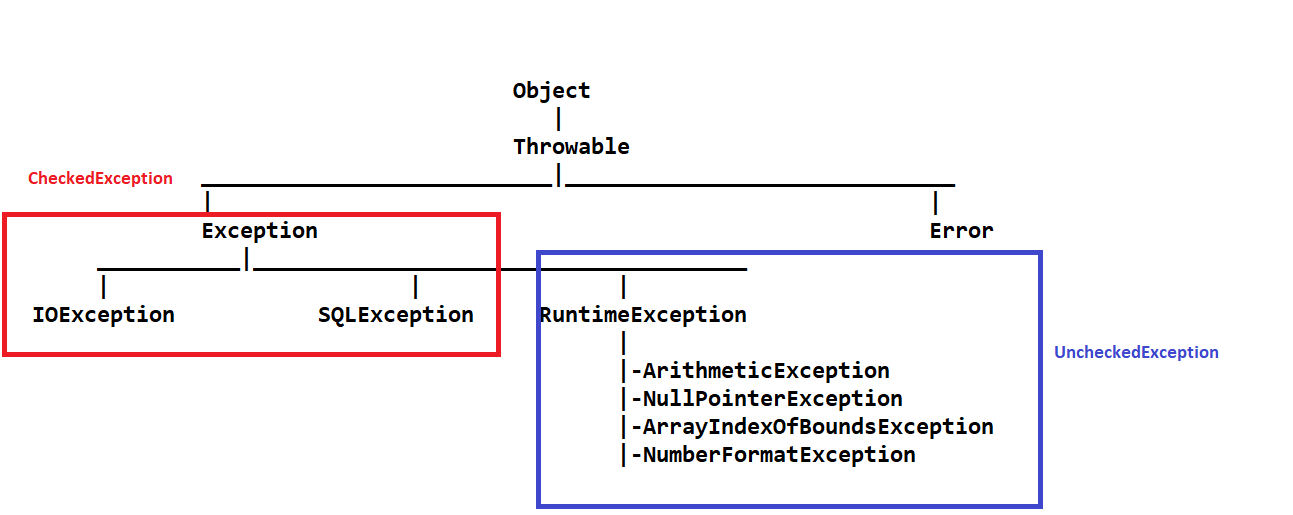
It is used to handle runtime errors, so that program doesn’t abruptly fail, we have five keywords in Java while handling exceptions

1. try: It is a block of code that might generate exception
2. catch: It is a block of code that can catch exceptions, you can have multiple catch blocks
3. finally: It is a block of code that is executed definitely after try/catch
4. throw: It is used to manually create & throw exception on certain condition
5. throws: It is used to propagate the exceptions to the caller so that he can decide what to do when the exception occurs

There are two types of exceptions

1. Checked Exceptions: It must be handled at compilation time itself
2. Unchecked Exceptions: It need not to be handled at compilation time itself

Exception Hierarchy



CheckedExceptions you had to handle, but UncheckedException need not to be handled, but they still can abruptly stops the application if it occurs

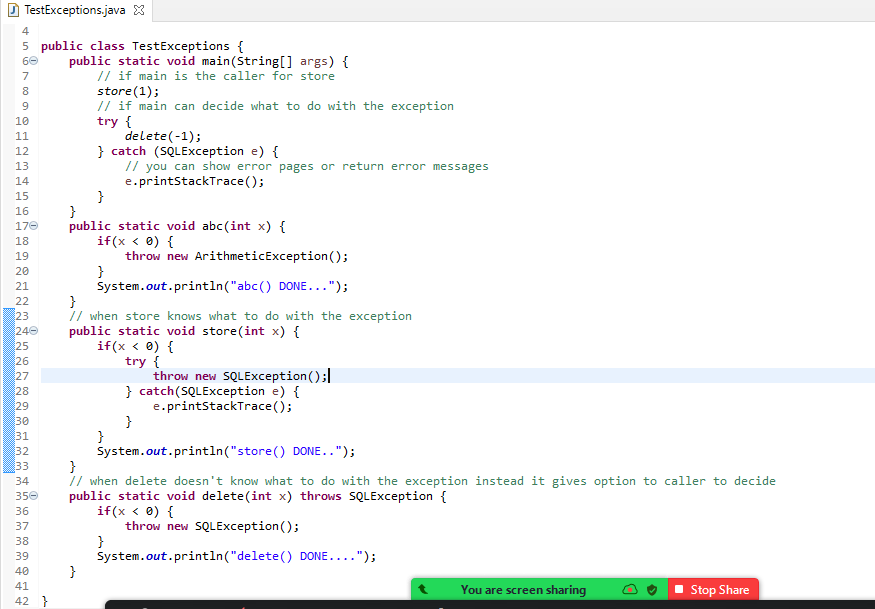
public void abc() throws ArithmeticException {   
  
}

When you call abc() you may expect ArithmeticException

public void store() throws SQLException {   
  
}

When you call store() you may expect SQLException

Note: Compiler doesn’t give any error if you don’t use try catch while calling abc(); However compiler gives error if you don’t use try / catch while calling store(), because it is propagating SQLException which is a checked Exception



Here store() itself handles exception, however the delete() doesn’t know how to handle the exception hence it propagates to the caller, the caller main or you can think caller as a view layer who handles exception and prints appropriate message/error message

Here we are generating predefined exceptions, but you can create your own exceptions by extending any of the Exception class, you can create checked or unchecked exceptions

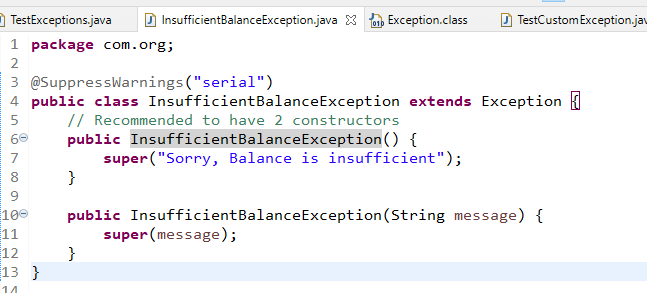
1. If you extend Checked exceptions like Exception, IOException, SQLException then your custom exception class will be checked exception
2. If you extend Unchecked exceptions like RuntimeException, ArithmeticException then your custom exception class will be unchecked exception

Suppose:

class InsufficientBalanceException extends Exception { } // checked exception

class InsufficientBalanceException extends RuntimeException { } // unchecked exception

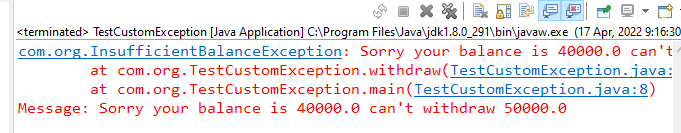
InsufficientBalanceException.java



TestCustomException.java



Output:



Activity:

1. Create a class that stores employee instances in the HashSet like a temporary database, store some employee objects, and find the employee by passing the id of the employee, if the id is not present throw a checked exception called EmployeeNotFoundException that shows “sorry employee with an <<id>> not found”, you have to ensure duplicate employee wouldn’t be stored in the HashSet.
2. Create your own example that uses all the OOPS concepts, Any Collection Framework class, LocalDate, & Exception Handling