Software requirement

1. Java 17 / JDK 17
2. Eclipse IDE

Java: Platform independent & Object Oriented Programming language

Platform independent: It can run java programs on multiple platforms without recompiling

Object Oriented Programming: You create real world entities in the application & these real world entities are called as objects, an object would interact with another object to complete its task

Object will have two things

1. properties: what it has
2. behaviors: what it does

Customer: customer\_id, name, dob, phone, emailId are the properties

deposit(), withdraw(), getDetails() are the behaviours

Employee: employee\_id, name, salary, phone

blockCards(), getCustomerDetails(), registerCustomer()

Building blocks of an object oriented programming language

1. class: It is blueprint/template for an object
2. object: It is an instance of the class

How to write java programs

You create java files & write java classes in it

class Employee {   
 properties : int id; String name; double salary;   
 behaviors : void registerCustomer() { …. } void blockCards() { ….. }  
}

Eclipse: It is an IDE (Integrated Development Environment) which speeds up the development

Workspace: It is a location where all the projects will be saved from eclipse

Java Fundamentals

1. datatypes
2. operators
3. conditional statements
4. loops
5. arrays
6. classes & objects

Datatypes: These are used to specify the type of data a variable stores

|  |  |  |
| --- | --- | --- |
| Datatypes | Size in bytes | default value |
| byte | 1 | 0 |
| short | 2 | 0 |
| int | 4 | 0 |
| long | 8 | 0 |
| float | 4 | 0.0 |
| double | 8 | 0.0 |
| char | 2 | ‘’ (empty character) |
| boolean | 1 | false |

byte, short, int & long stores whole numbers  
float & double stores floating point numbers (real numbers)  
char: to store single character, like M, F, Y, N  
boolean: true or false

How to declare variables:

Syntax: datatype variable = value;

ex:   
int id = 1234;  
double salary = 50020.0  
char gender = ‘M’; // char must use single quote only, double quotes are used in String  
boolean isAdmin = true;

Operators in Java

Arithmetic operators: +, -, /, \*, =

Comparison operators: <=, >=, ==, !=

Logical operators: &&, ||

Increment & Decrement operators: ++, -- (double minus)

Arithmetic operators are useful incase of calculations

double amount = 5000;  
double balance = 15000;  
balance = balance – amount;

double price = 1000;  
double discount = 0.1;  
double total = (price – (price \* 0.1)) \* 1.18; // (1000 – 100) \* 1.18 = 900\*1.18 = 1062

Comparison operators are useful incase of comparisons

int id = 1234; // id gets the data from the UI  
int employeeId = 1234; // employeeId gets the data from DB  
boolean id == employeeId; // id = true

Logical operators are useful to compare multiple expressions

int x = 10;  
int y = 20;  
int z = 30;  
boolean r = (x < y) || (x > z); // 10 < 20 || 10 > 30 // r = true  
boolean s = (x < y) && (x < z); // 10 < 20 && 10 < 30 // s = true

Increment & Decrement operators are useful in case of increasing or decreasing the values

int counter = 0;

counter++; // counter = 1  
counter++; // counter = 2

int anotherCounter = 15;  
anotherCounter--; // anotherCounter = 14

Post and Pre operations in case of increment & decrement

Post increment/decrement: Assigns the value and then increments/decrements

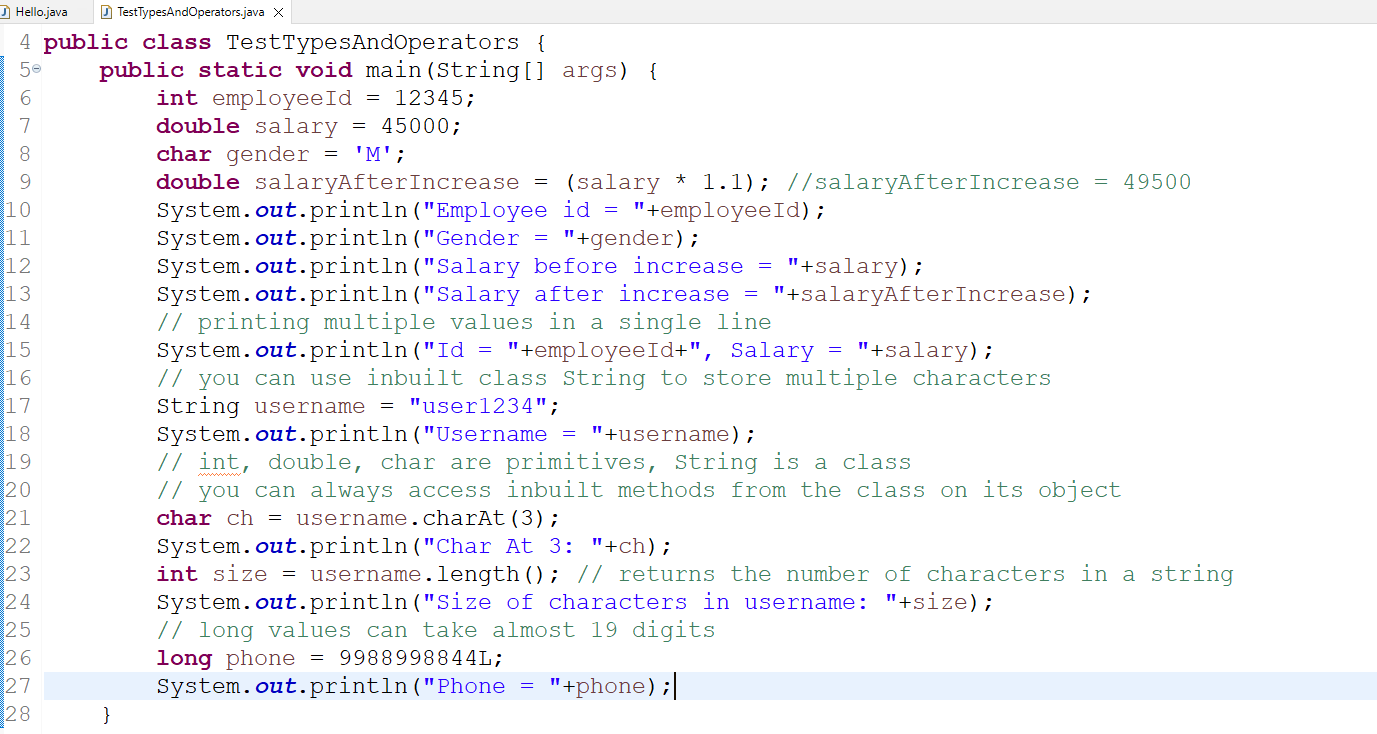
Pre increment/decrement: Increases/Decreases the value and then assigns

int a = 5;  
int b = a++; // b = 5, a = 6

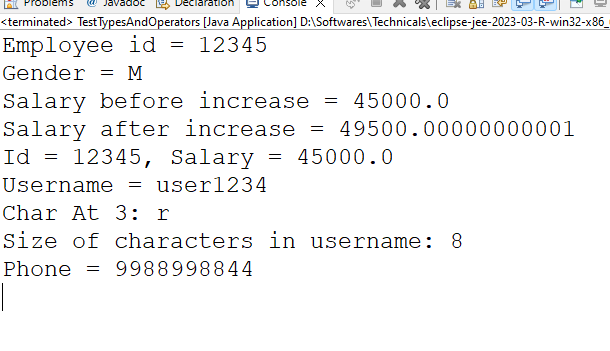
int c = 5;  
int d = ++c; // c = 6, d = 6

counter = 0;  
employeeId = ++counter; // employeeId = 1,

TestTypesAndOperators.java



Output:



Scanner class: It is present in java.util package, which must be imported to use that class, it allows user to give dynamic values from the keyboard / console

Note: Classes from java.lang are auto-imported, classes like String, System, Math need not import

import java.util.Scanner; // if you use eclipse it auto-imports when you use control + space

Java file structure

package statement; // only one package statement in one java file, it must be at the top

import statement; // comes after the package statement, you can zero or more imports

class className { // comes after all the imports

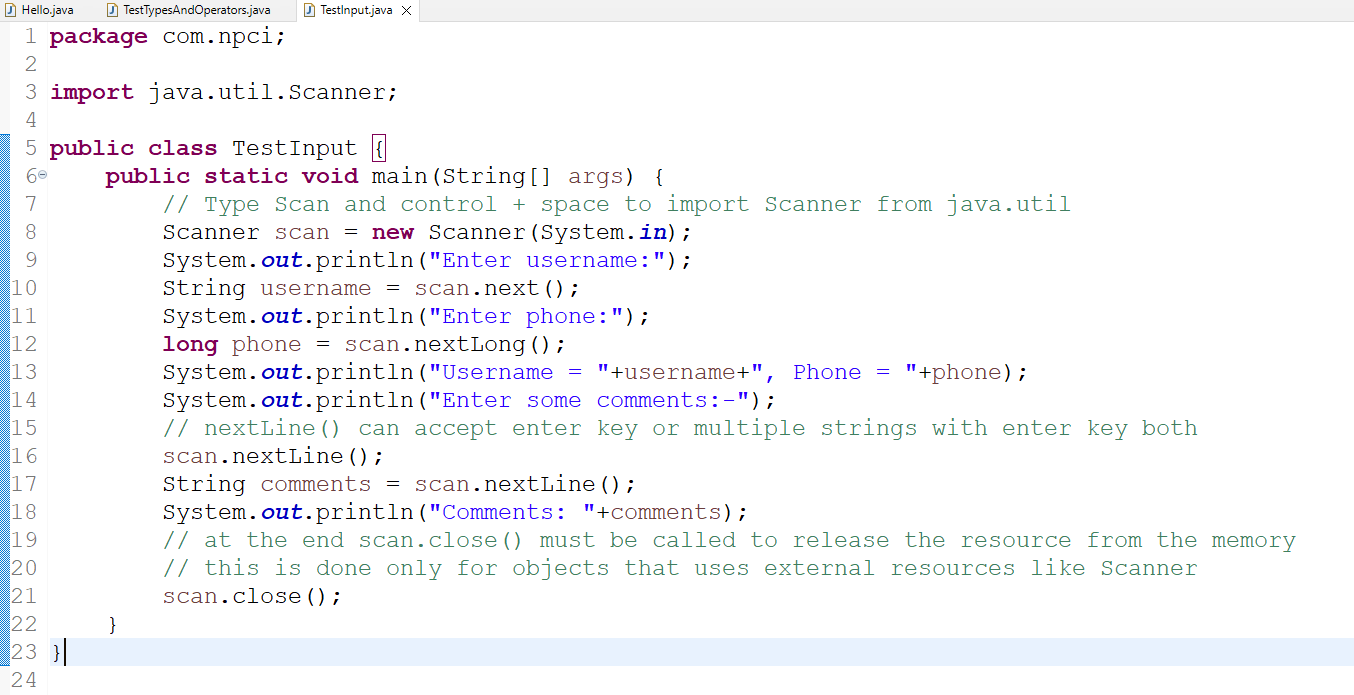
import java.util.Scanner;

We need to create Scanner object to access their methods

Scanner scan = new Scanner( System.in );

scan.nextInt(); // to take int value  
scan.nextDouble(); // to take double value  
scan.next(); // to take a single string  
scan.nextLine(); // to take multiple strings until you press enter key

TestInput.java



Conditional Statements

These are used when you want to run certain block of statements when certain conditions are met.

1. if
2. if else
3. if else if else if … else
4. switch

If:

statement  
if(a > b) {   
 statement  
}

If and Else:

statement  
if(a > b) {   
 statement  
} else {  
 statement  
}  
statement

If, Else If & Else:

statement  
if(a > b) {   
 statement  
} else if (a > c) {   
 statement  
} else if ( a > d) {   
 statement  
} else {   
 statement: executed if all the above conditions are false  
}

switch:

switch(options) {   
 case value1: statement;  
 break;  
 case value2: statement;  
 break;  
 default : statement;  
}

Ex:

marks = scan.nextDouble();  
if(marks >= 70) {   
 // A+  
} else if (marks < 70 && marks >= 60) {   
 // A  
} else if (marks < 60 && marks >= 50) {   
 // B  
} else { // C }

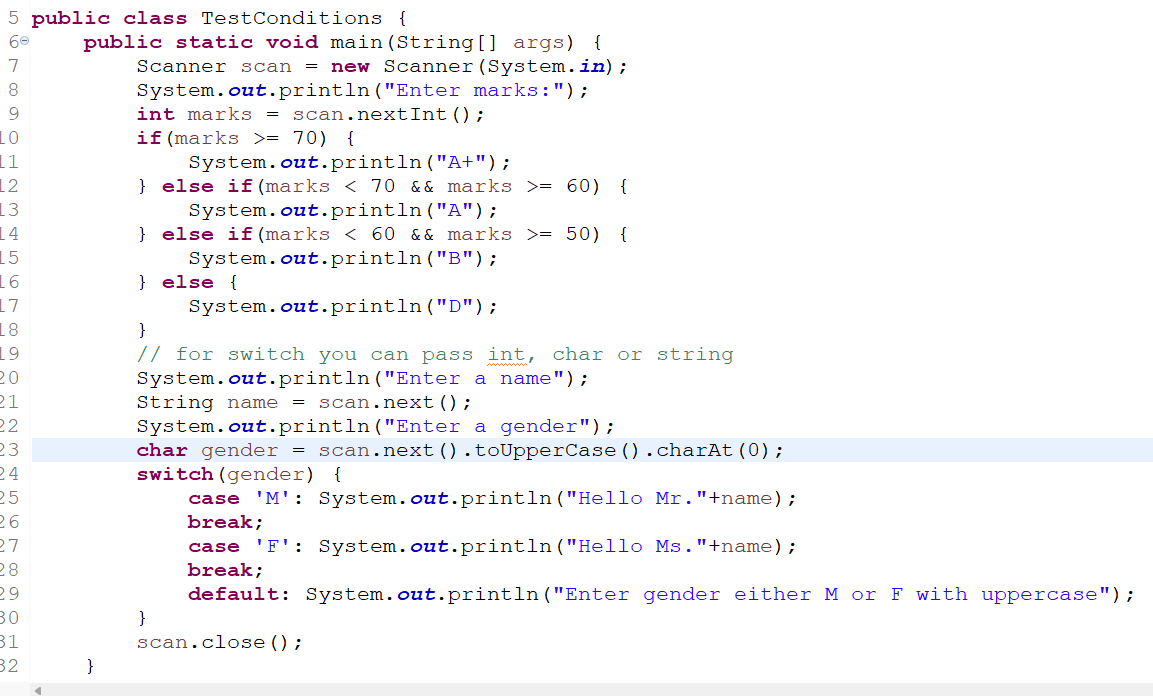
Ex:

option = scan.nextInt();

// switch can compare int, String, char   
switch(option) {  
 case 1: //statements; break;  
 case 2: // statements; break;  
 case 3: // statements; break;  
 default: // statement

}

TestConditions.java



Loops in Java

When you want to run some statements repeatedly until some condition is true you can use loops, we have 3 loops in java

1. for loop
2. while
3. do while

Array: It is a collection to store multiple values  
int[] items = { 3, 1, 4, 5, 2}  
print items[0]; // prints 3  
print items[2]; // prints 4  
print items.length; // prints 5

for loop: When you want to iterate for fixed number of items

for(int index = 0; index < items.length; index++) { // index = 0, 1, 2, 3, 4, at 5 it exits  
 print items[index]; // 3, 1, 4, 5, 2  
}

while loop: When you want to iterate until some condition is true, it first checks the conditions & then executes the loop

do while loop: Similar to while loop, but atleast once the loop is executed & then the condition will be checked

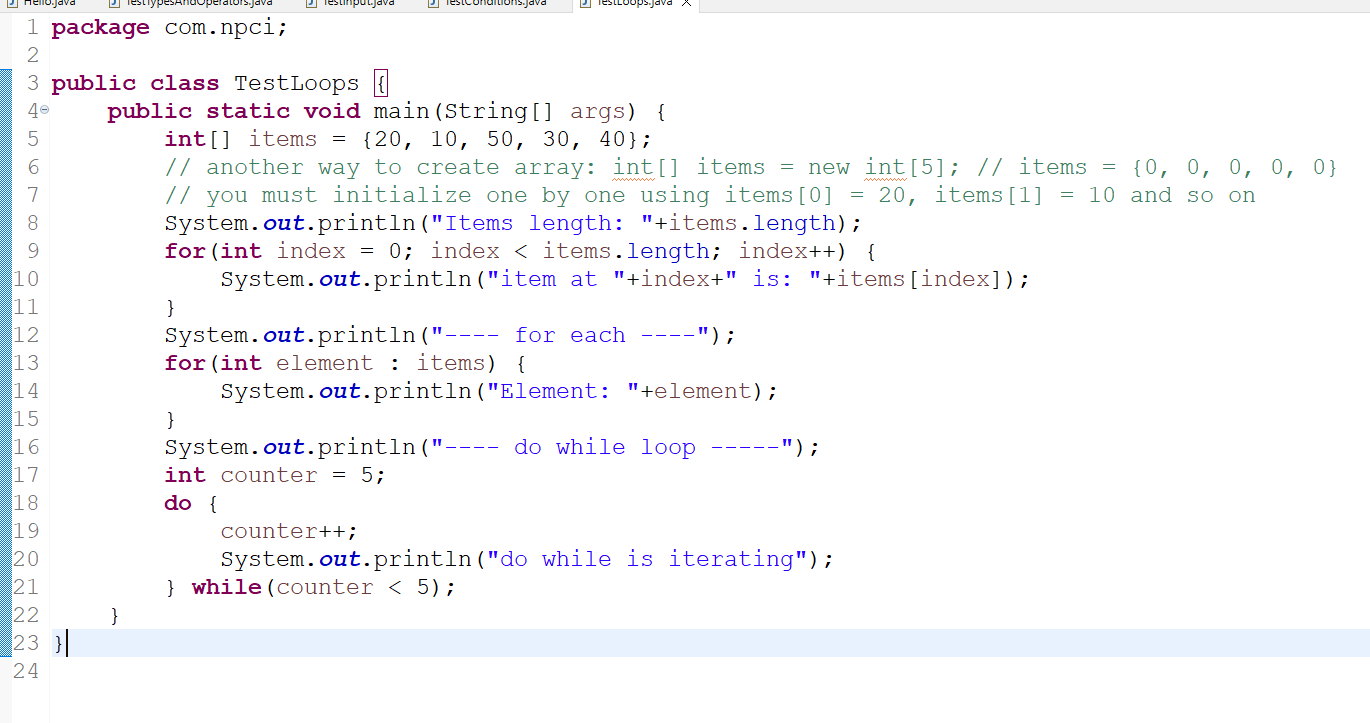
counter = 0;  
while(counter != 5) {   
 counter++;  
}  
counter2 = 5;  
while(counter2 != 5) {  
 counter2++;  
}  
counter3 = 5;  
do {   
 counter3++;  
}while(counter3 < 5);

Note: counter3 becomes 6 because do while will be executed once.

Enhanced for loop / for each: Simplified for loop used on arrays & collections  
Syntax:  
for(type t : collection) {   
 // t is an iterator variable that gets the value from the array  
}

String[] items = {“hello”, “welcome”, “thankyou”}  
for(String s : items) {  
 print s  
}

TestLoops.java



Activity

1. Create an array of some numbers and print their sum of numbers in the array, then maximum and minimum number in the array, use only one loop to perform all the task

ex: if items = {10, -3, 4, 3, 9, 8}, Sum = 31, Max = 10, Min = -3

1. Enter 3 digits number and print each digit in words

ex: Input = 472, Output = Four Seven Two

1. Enter 3 digits number and multiply the highest digit number with the lowest digit number.

ex: Input = 472, Output = 7 \* 2 = 14

Command line argument:

The String[] parameter in the public static void main(String[] args) is the command line argument which can accept the input you pass from the command line before executing the program

These arguments can take input before starting the program, so that the program can use that input and start some servers or connect to some database

java Hello arg1 arg2 arg3 arg4

All these arguments are stored in String[] args (it is optional to pass arguments) but when you use them in some advanced technologies they give lot of benefits

ex: In Spring boot you can pass command line arguments with server port, database credentials as an argument, so that the program can use those command line arguments.

java Hello –server.port=9091 –database.username=root –database.password=12345

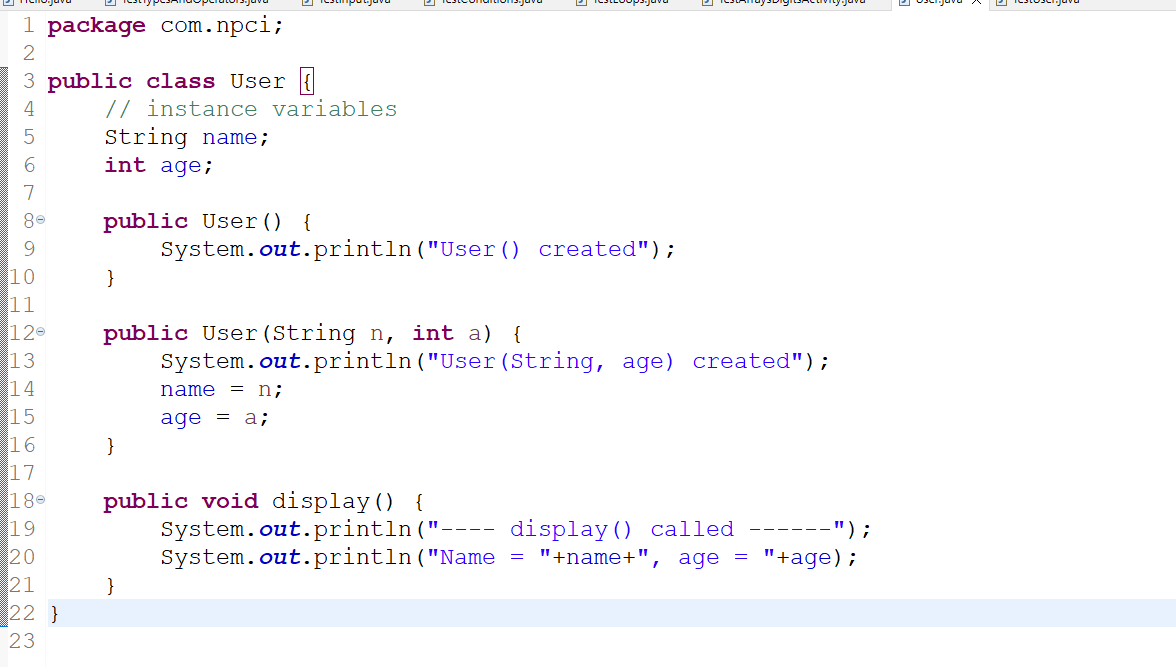
String[] args = {“-server.port=9091”, “-database.username=root”, “database.password=12345” }

classes & objects

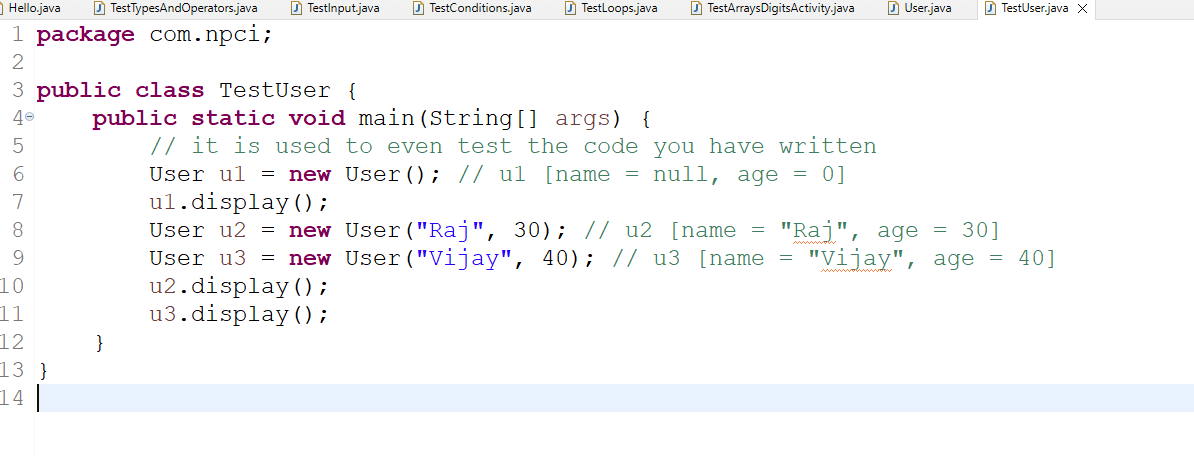
A class can have following things

1. variables
2. methods
3. constructors

User.java



TestUsers.java



Summary:

1. javac & java commands
2. fundamentals – operators, datatypes, arrays, loops, conditional
3. command line argument
4. class & objects

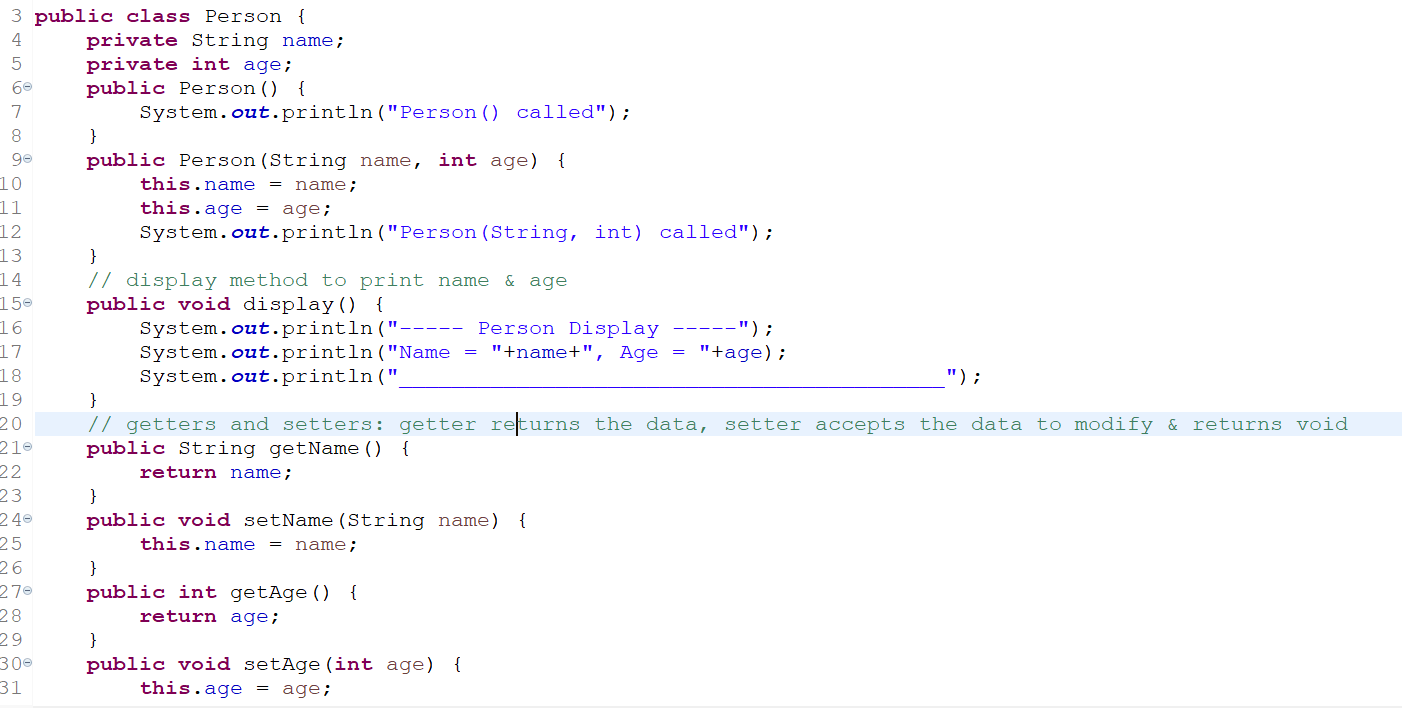
OOPs features

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

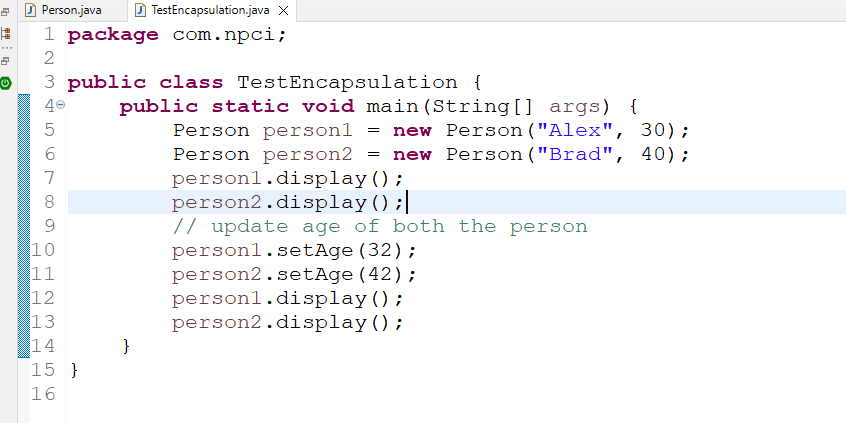
Encapsulation:

Hiding the data(private variables) where you will have more control over the data, you can’t directly access these data outside the class, however you can access them using public methods of the class

Person.java



TestEncapsulation.java



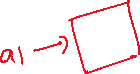
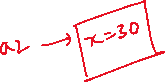
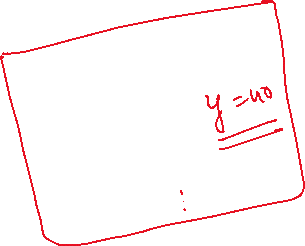
static members:

These are the members which you can access without creating the object, you can create

1. static variables and
2. static methods

both can be accessed without creating object with the use of class name

Ex:  
class A {   
 int x = 30; // instance variable  
 static int y = 40; // static variable  
}  
System.out.println(A.y);



A a1 = new A();  
A a2 = new A();  
System.out.println(a1.x);   
System.out.println(a2.x);



static variables: All the objects share single copy of the static variables

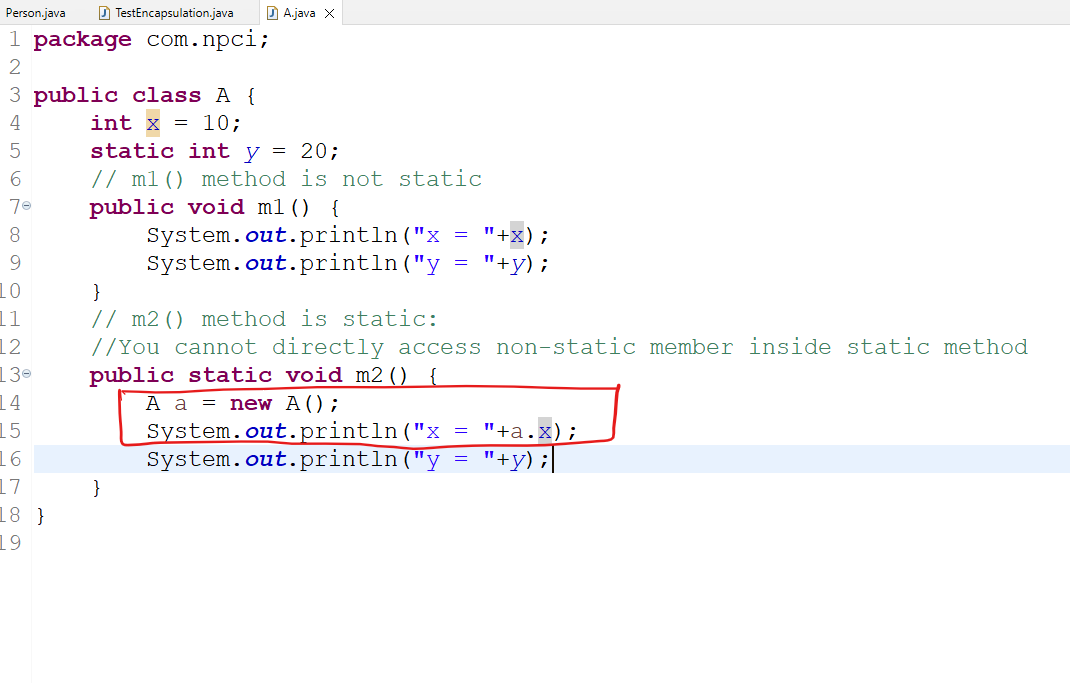
instance variables: Every object will have its own copy of the instance variable

static methods: It can be called directly using the class name, when you want some operations to be called without creating object or some common operations for all the objects then you can use static methods

ex: public static void sort() { // want to sort all the objects }  
ex: public static void main(String[] args) { this method is available before any object is created }

Understanding what we can access from a static method & what not

A static method cannot directly access the non-static member, if required it can use the reference of the object and access the non-static member.

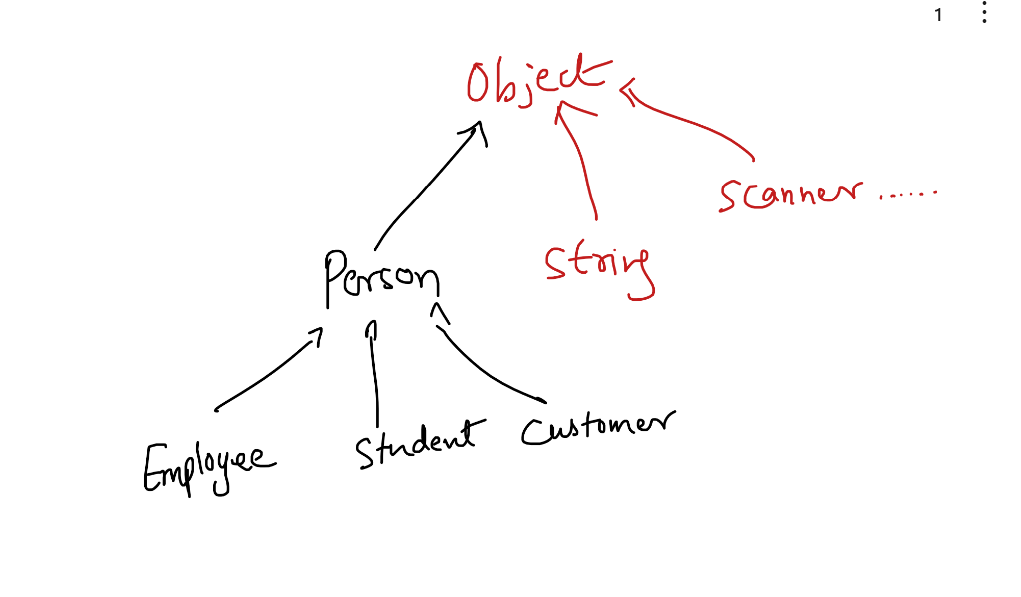


Inheritance

It is used to acquire the properties & behaviors of a class from another class, you can create common features in the parent class/super class/base class and inherit in the child class/sub class/derived class

ex:   
class Person { name & age }   
class Employee extends Person { id, salary }   
class Student extends Person { id, salary }

Note: Every class will be a subclass to the Object class(inbuilt class) either directly or indirectly



Note: Every subclass constructor automatically calls its parent class default constructor, this is mainly to initialize the super class members before initializing the subclass members, if the parent class doesn’t have default constructor then subclass must explicitly call the overloaded constructor of the parent class else compiler raises error, because compiler is the one who automatically writes the statement to call the parent class default constructor.

class A {   
  
}



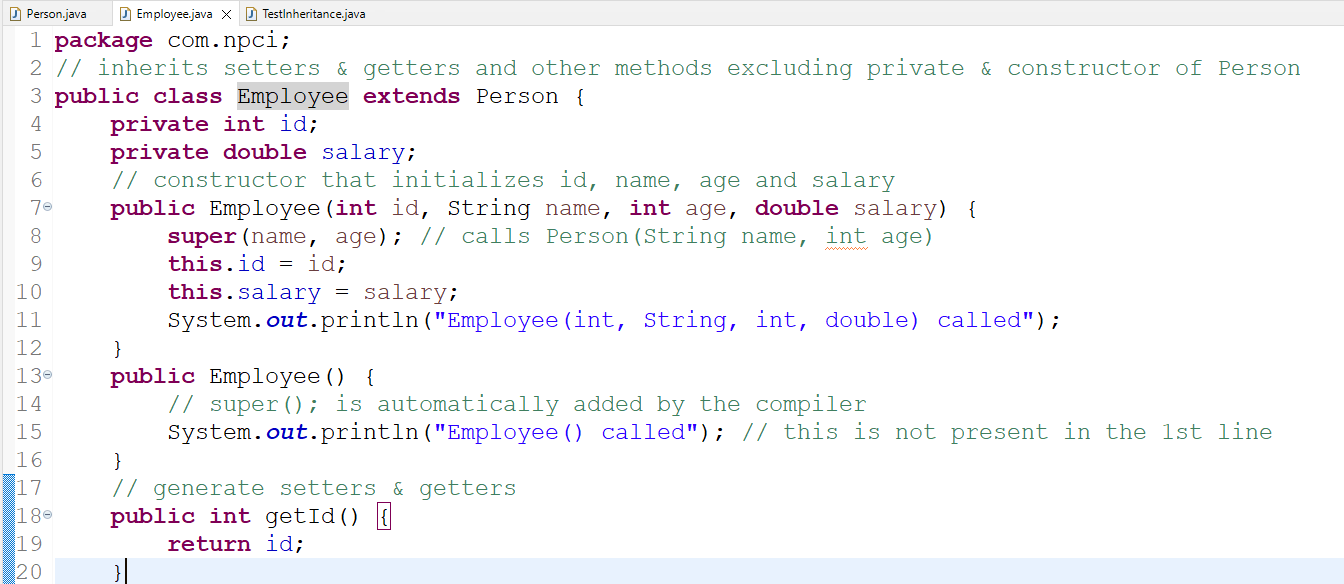
class B {   
 B() { stmts; }   
 B(int x) { stmts; }  
}



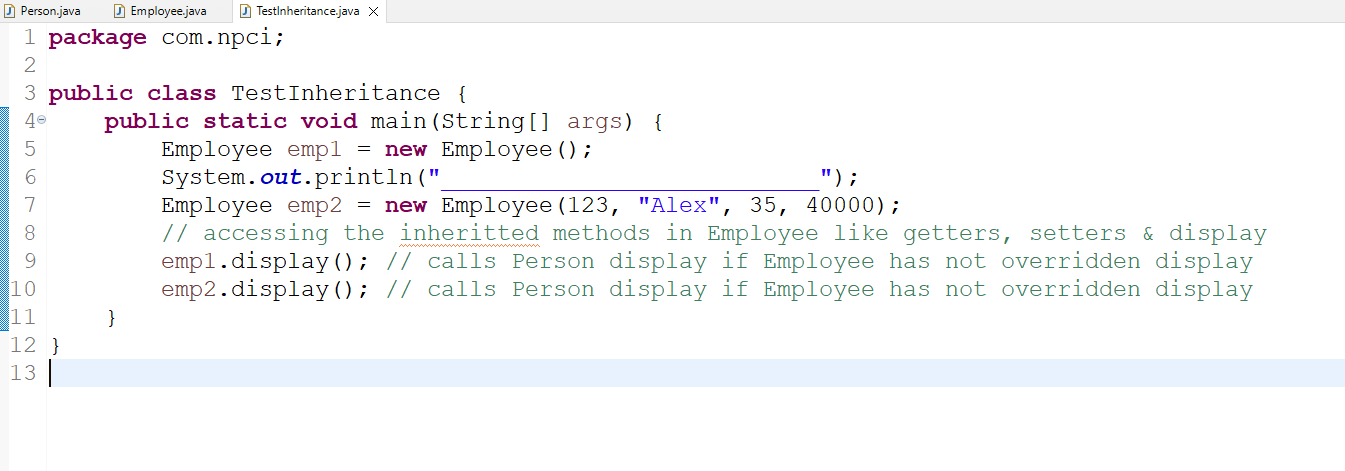
class C {   
 C(int x) { }   
}  
class D extends C {   
  
}  
class E extends C {   
 E(int x, int y) { super(x); stmts; }  
}



Employee.java

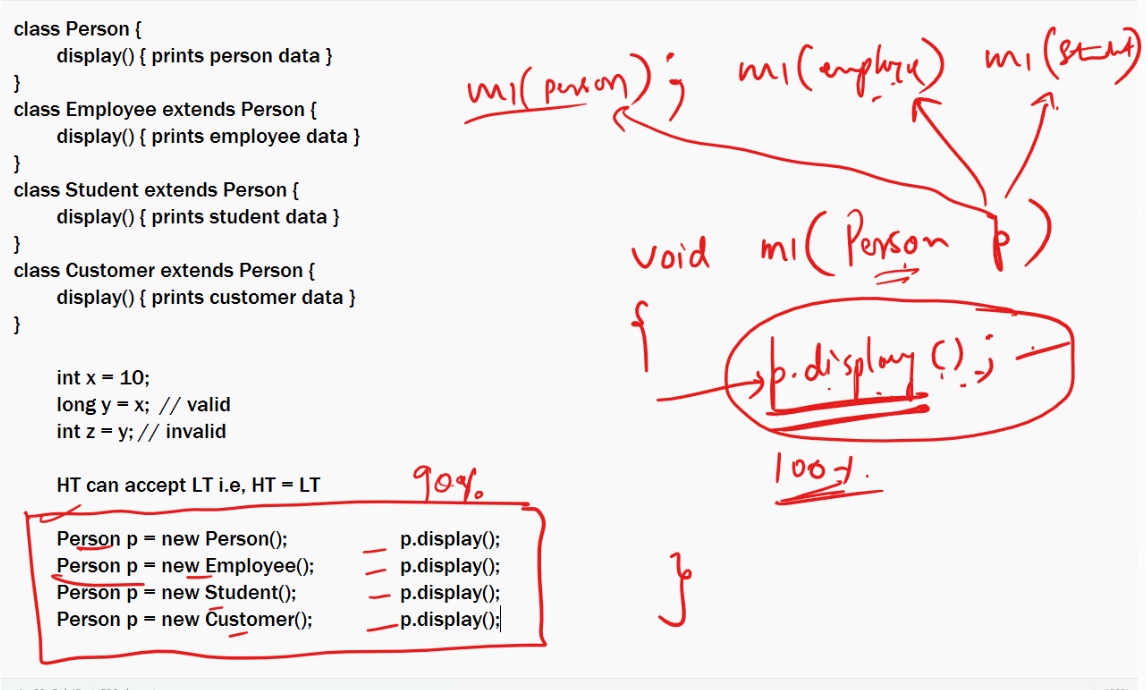


TestInheritance.java



Note: display() method is called on Person class because its not overridden

Polymorphism: A method with many actions



Method Overriding: Having same method names and signature but different implementations in the subclass

To achieve polymorphism we need to override methods and have a higher type to call this method, so that the higher type can accept all the objects of its lower type and when the reference of higher type calls the overridden method it calls on the multiple objects

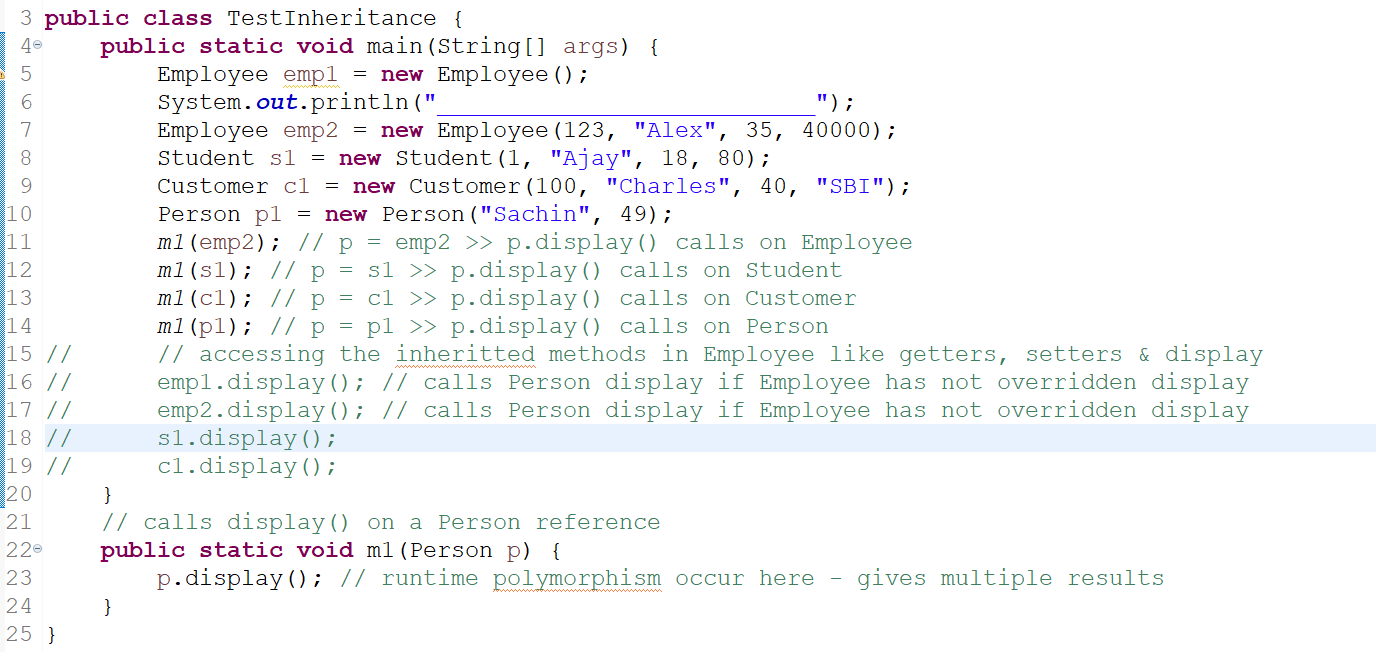
Activity:

Create 2 subclasses of Person

1. Student: will have rollNo, marks
2. Customer: will have customer\_id and bankName

Create Student object & Customer object by passing their respective data including name & age and on each object call the display (now no need to override display in the sub-class)

Testing polymorphism



Activity:

Create an array of Person as Person[] p = new Person[size], where size would be taken at runtime, which specifies how many objects it can store, each index of Person[] can store any sub class object of Person, show a menu option in a loop until you wish to exit, the menu option will be   
1: Student 2: Employee 3: Customer 4: Display All 5: Exit  
When you enter 1: Ask for student details like rollNo, name, age & marks then store the object in the Person[], similarly when you enter 2: Ask for employee details and store in the Person[], then when you enter 3: Ask for customer details & store in the Person[].

When you enter 4: iterate the Person[] and pass the iterated object to the m1(Person) method implemented in the TestInheritance which is already calling display.

Repeat the process until you enter 5.

Abstraction: Hiding the complexity from the user & showing only the necessary details, it can be achieved in two ways

i.e., hiding the implementation and showing only the method signature

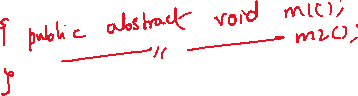
1. interface
2. abstract class

interface: It is like a class but will have only abstract methods (methods without body) & constants

ex:



interface X {   
 void m1();  
 void m2();  
}



Note: interface methods are by default public & abstract  
X interface must be implemented by some developer using class

class ImplOne implements X {   
 // class must implement all the abstract methods of the interface  
}  
class ImplTwo implements X {   
 // class must implement all the abstract methods of the interface  
}

Important points on interface

1. Interfaces can’t have constructors, not even default constructor
2. You can’t create object of interface, but you can always use them as a reference variable

X x1 = new ImplOne()   
X x1 = new ImplTwo();

1. Interface can also hide methods from the object which are restricted to access

ex:   
interface X {   
 void book();  
}

interface Y extends X {   
 void modify();  
}  
class IRCTC implements X, Y {   
 void book() { … }   
 void modify() { … }  
}  
Regular User >>> m1(X x) { x.book() } >> can call only book method because he has X interface

Admin User >>>> m2(Y y) { y.book & y.modify } >> can call both the methods because he has Y interface

Abstract class

These are the classes that can have abstract methods & methods with body, when you don’t know logics of all the methods then you can use abstract class

abstract class Car {   
 void wheels() { prints 4 }  
 abstract void mileage();   
}  
Note: abstract classes must use abstract keyword mandatorily for abstract methods, however in interface they are abstract by default.

class Nexon extends Car {   
 void mileage() { prints 18kmpl }  
}  
class I20 extends Car {   
 void mileage() { prints 22kmpl }   
}

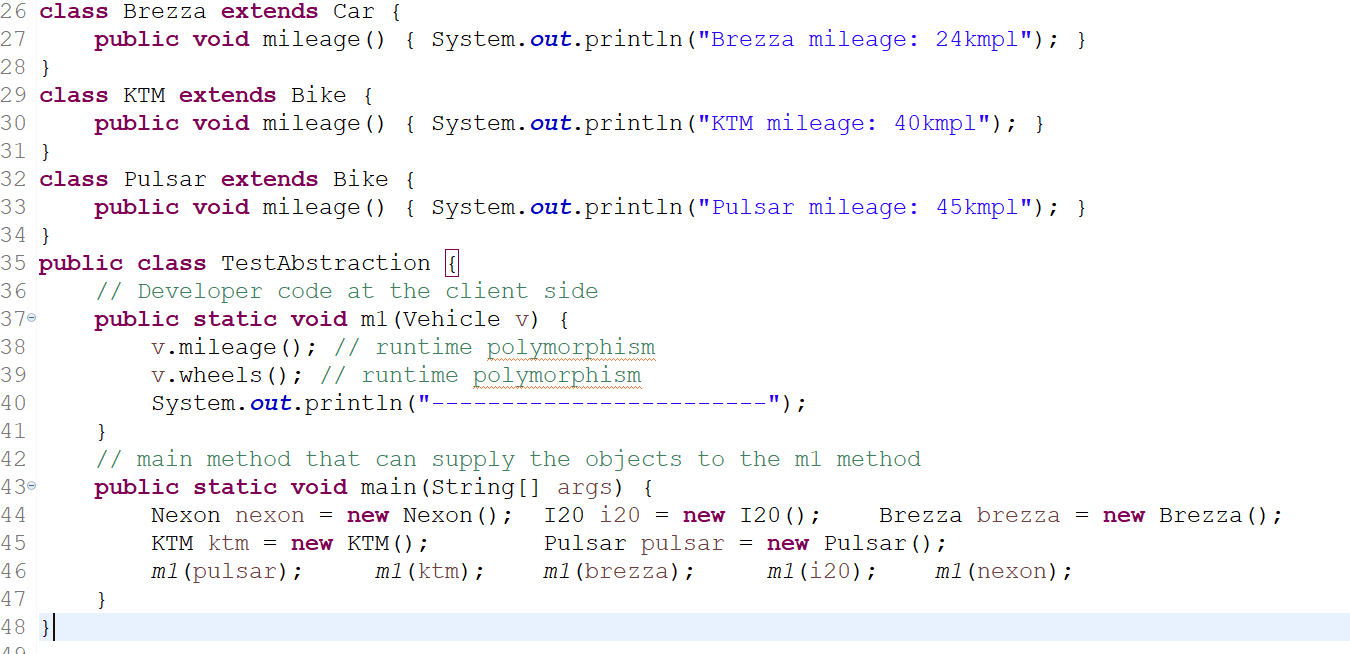
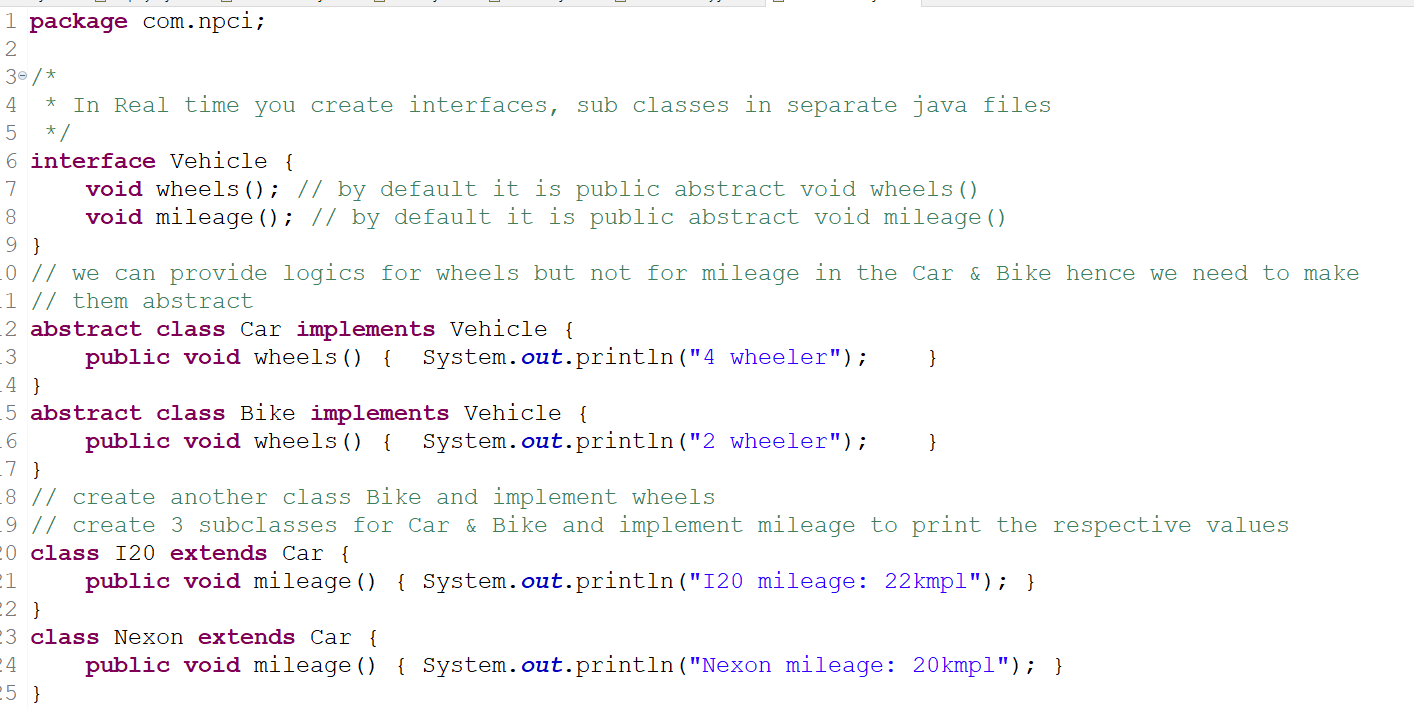
Important points on abstract class

1. You can’t create object of an abstract class
2. You can have constructors to initialize the properties, however abstract keyword restricts object creation & these constructors are called by subclass object

Constructor chaining: calling a constructor from another constructor within the class, using this()

A(int x, int y) {  
 this.x = x;  
 this.y = y;  
}  
A(int x, int y, int z) {   
 this(x, y) // A(x, y) // when this(args) is used super() is not called  
 this.z = z;  
}

TestAbstraction.java



Exception handling

Exceptions are runtime errors that abnormally terminates the program if not handled, there are 5 keywords we use in exception handling

1. try
2. catch
3. finally
4. throw
5. throws

try: It is a block which can have the codes that may generate an exception

catch: it is a block which will handle the exception, you can have one or more catch blocks

finally: it is a block that is definitely executed regardless of whether exception is handled or not or exception occurred or not occurred

throw: It is used to manually generate the exception

ex:

if(amount > balance) {  
 throw new ExceptionName(“some exception message”);  
}

if(trasactionLimitExceeds == true) {   
 throw new ExceptionName(“some exception message”);  
}

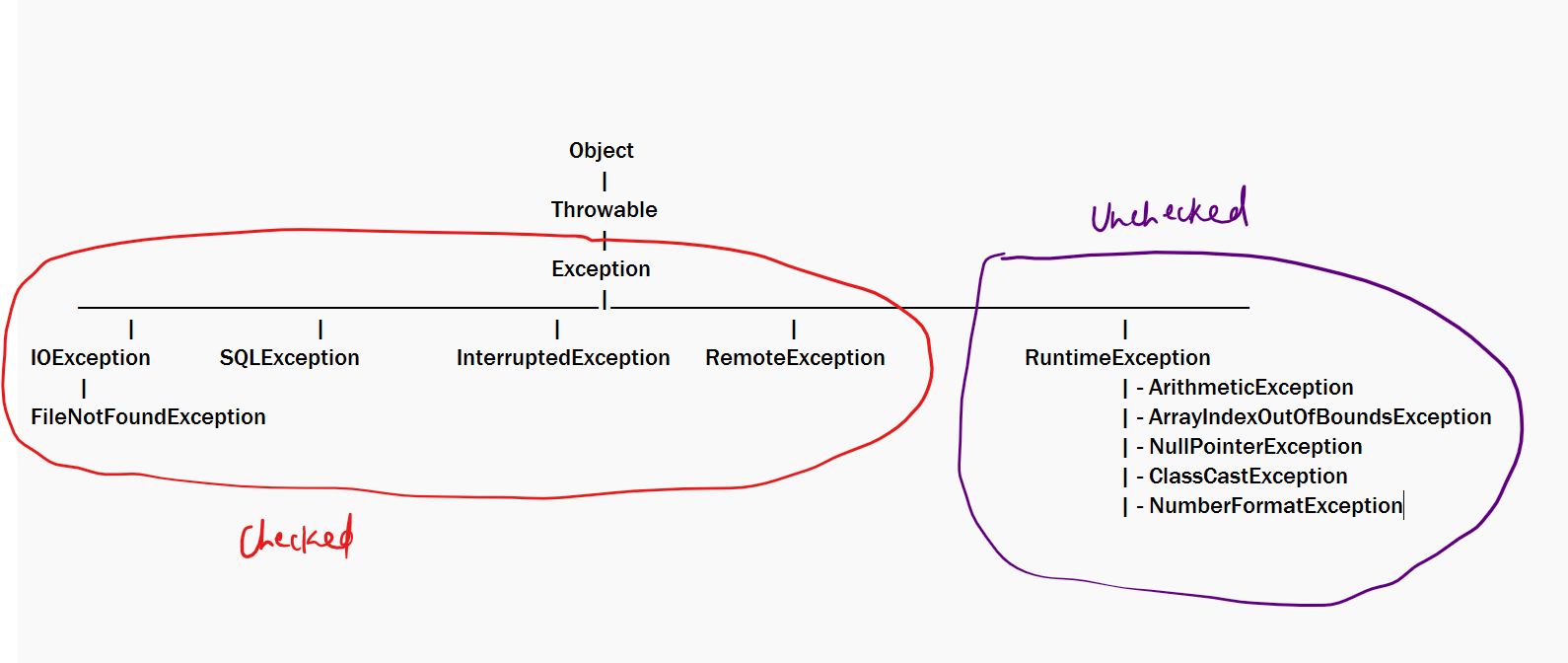
Note: throw keyword will generate the exception, but it must be handled or must be propagated to the caller

throws: It is a keyword used in the method signature to propagate the exceptions to the caller when a method doesn’t want to handle the exception

ex:

public double debit(double amount) throws TransactionDeclinedException {   
 // generate the exception but propagate instead of handling  
 if(amount > balance) throw new TransactionDeclinedException(“some message”)  
 // return the balance if no exception  
 return balance;  
}  
caller  
try {   
 debit(1000);  
 success response  
} catch(TransactionDeclinedException e) {   
 error response  
}

Exception Hierarchy



Checked Exceptions: These are checked at the compilation time and you are forced to handle them, if the exception occurs it will be already handled, if not occur then fine

Unchecked Exceptions: These are not checked at the compilation time & are not forced to handle them, however if an unchecked exception occurs and you have not handled then there might be an abnormal termination

Methods & Constructors usually use throws to specify what exceptions will be generated, when we call these methods we may or may not get exceptions however if they are using throws with unchecked exceptions compiler will ignore, but if they are using throws with checked exceptions compiler will force you to handle using try/catch

ex:

void m1() throws SQLException { …. }  
void m2() throws ArithmeticException { …. }

try {   
 m1();   
} catch(SQLException e) { … }

Note: You can’t call m1() unless you handle the exception with try/catch

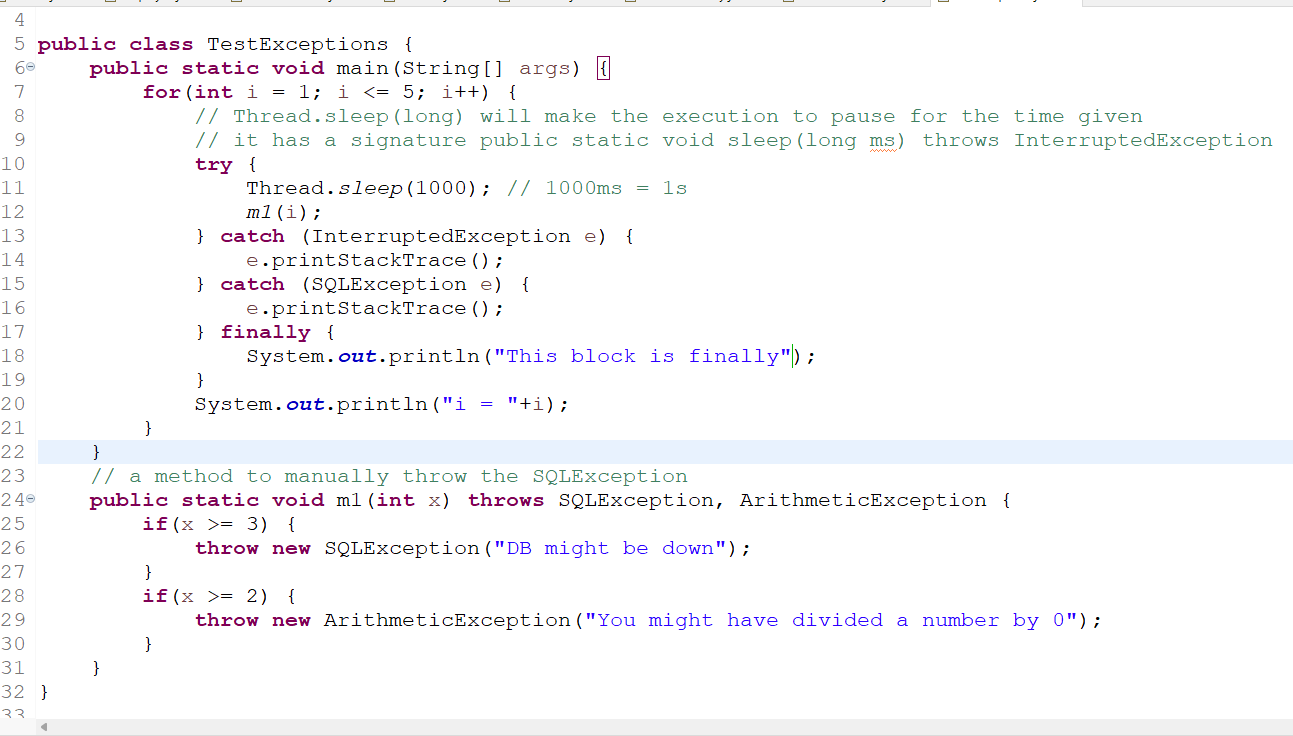
You can call m2() without try/catch because compiler will ignore unchecked exceptions

A method can also propagate more than once exception as below  
void m3() throws SQLException, RuntimeException, IOException { }

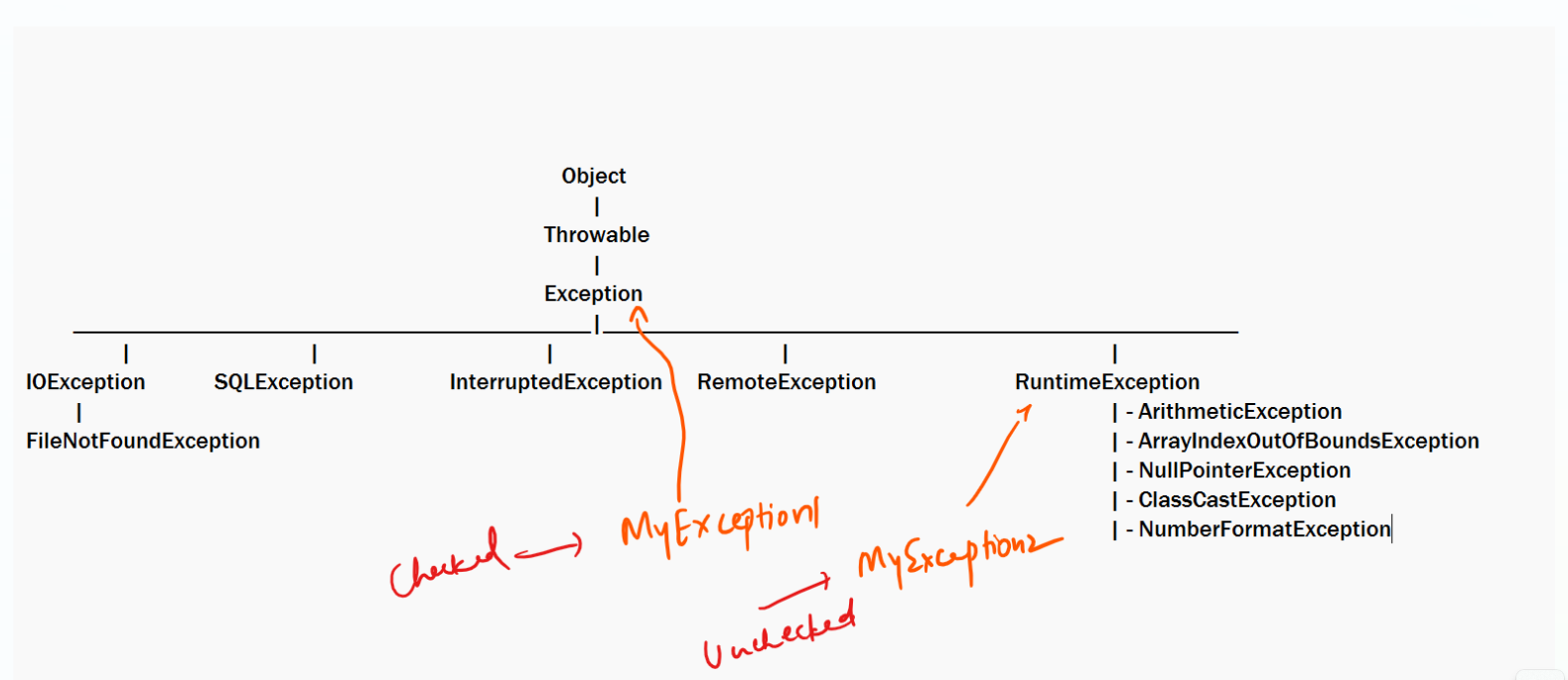
You must mandatorily handle checked exceptions like SQLException & IOException

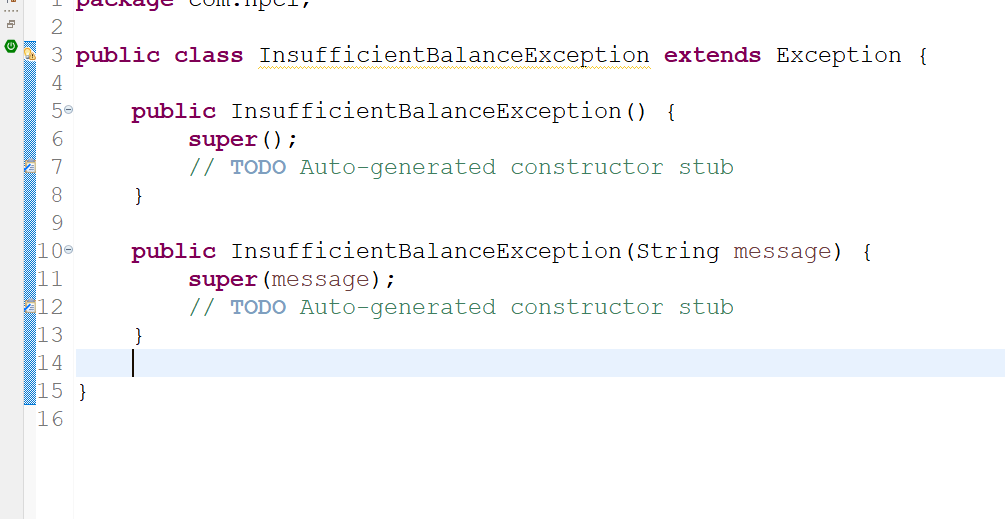
try { m3(); } catch(SQLException e) { } catch(IOException e) { }

TestExceptions.java

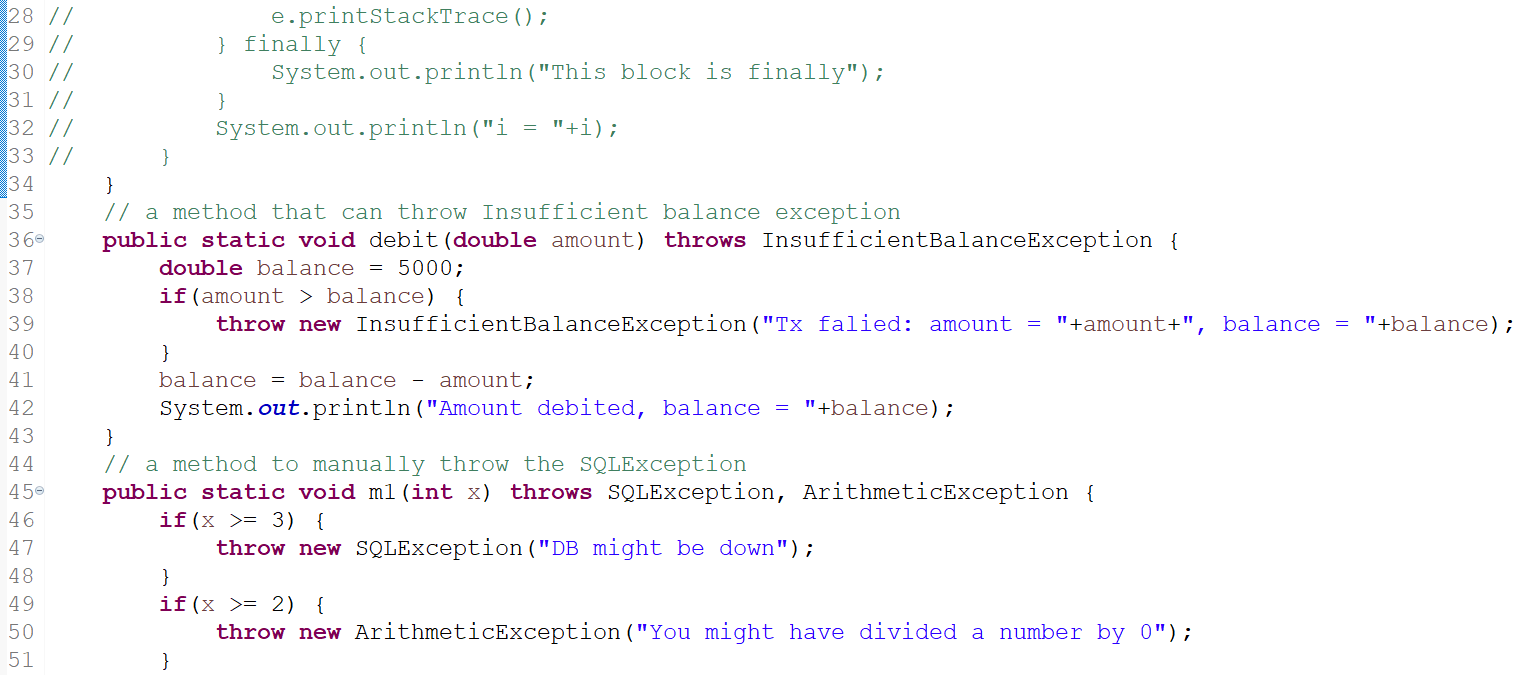
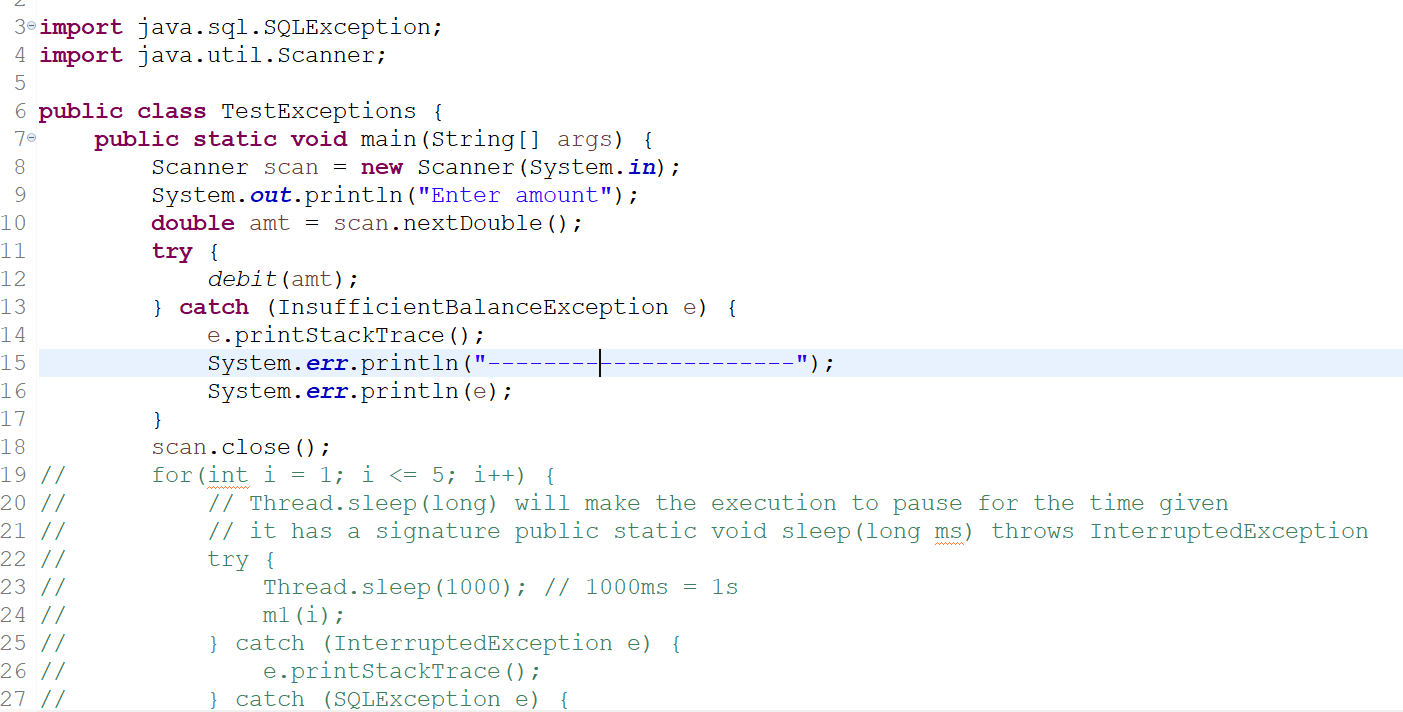


Custom exceptions: You can create your own exceptions by extending Exception or RuntimeException which can be checked or unchecked, but most preferred is checked exception.  
ex: public class InsufficientBalanceException extends Exception { … }  
Note: In custom exceptions also you need to create constructors that are used to generate exceptions





TestExceptions.java



Activity:

Create an interface with a name EmployeeOperation that will have following methods

1. void store(Employee employee) throws ArraySizeExceededException;
2. Employee findById(int id) throws EmployeeNotFoundException;
3. Employee[] findAll();  
   Implement the interface that will maintain the data in an array of some specific size, keep this implementation as a separate class, the methods like store(), findById(), findAll() must be called from the main method  
   Create a separate class for main method will have 4 options like:-  
   1. store 2: find by id 3: find all 4: exit  
   The above options must be in a loop until you enter 4.  
   on option 1: Ask for id, name & salary from scanner, create Employee object & pass that object to the store() method, the store() method will store in an Employee[], if the array is full throw the exception & print the error message in the main   
   on option 2: Ask for id, and call the findById() by passing the id, the findById() method must return the Employee matching to id or throw the EmployeeNotFoundException, in the main handle the exception & print the error message  
   on option 3: call findAll() which returns the Employee[], so that main method iterates and prints all the employee details

Note: Main method is like UI that will have logics of taking input, invoking the logics & printing output

Inbuilt classes

Object class: It is the super class for all the classes, many classes override some methods of Object class, which are:-

1. public String toString(): This method is automatically called when you print object, by default it returns the memory address, you can override to return object information in your class
2. public boolean equals(Object obj): This method compares two objects, it returns true if two objects are same else false, by default it compares two objects memory address but you can override to compare the object properties you want like id, customer\_id, rollNo and so on
3. public int hashCode(): This method returns the unique number for every object, it can also be overridden

Note: equals & hashCode we can understand when we use in Collection Framework

LocalDate: It is present in java.time package, used to maintain the date, it follows ISO standard to maintain the date (“yyyy-MM-dd”)

LocalDate today = LocalDate.now(); // stores the current date  
LocalDate another = LocalDate.parse(“1970-10-25“);   
System.out.println(today); // internally calls today.toString() & prints 2024-03-27

String: It creates an immutable string object

s1.equals(s2): compares the content  
s1.concat(“123”): creates a new string after concatenation

String text = “123;Raj;35000”  
String[] str = text.split(“;”); // str = {“123”, “Raj”, “35000”}

converting string to int or double

int x = Integer.parseInt(str[0]); // convert “123” to 123 in number format  
int y = Integer.parseInt(str[1]); // can’t convert Raj to number, hence you get NumberFormatException

double z = Double.parseDouble(str[2]); // converts “35000” to 35000 in double format.

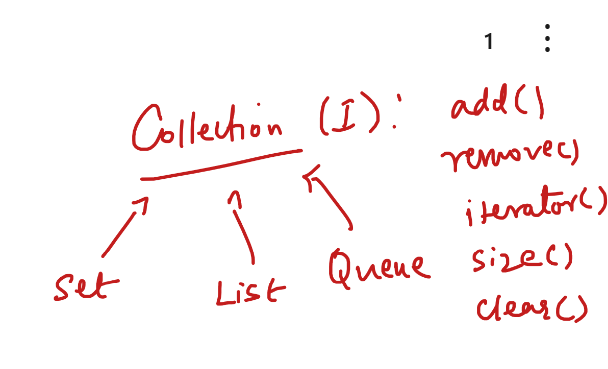
Comparing two values using compareTo or compare methods

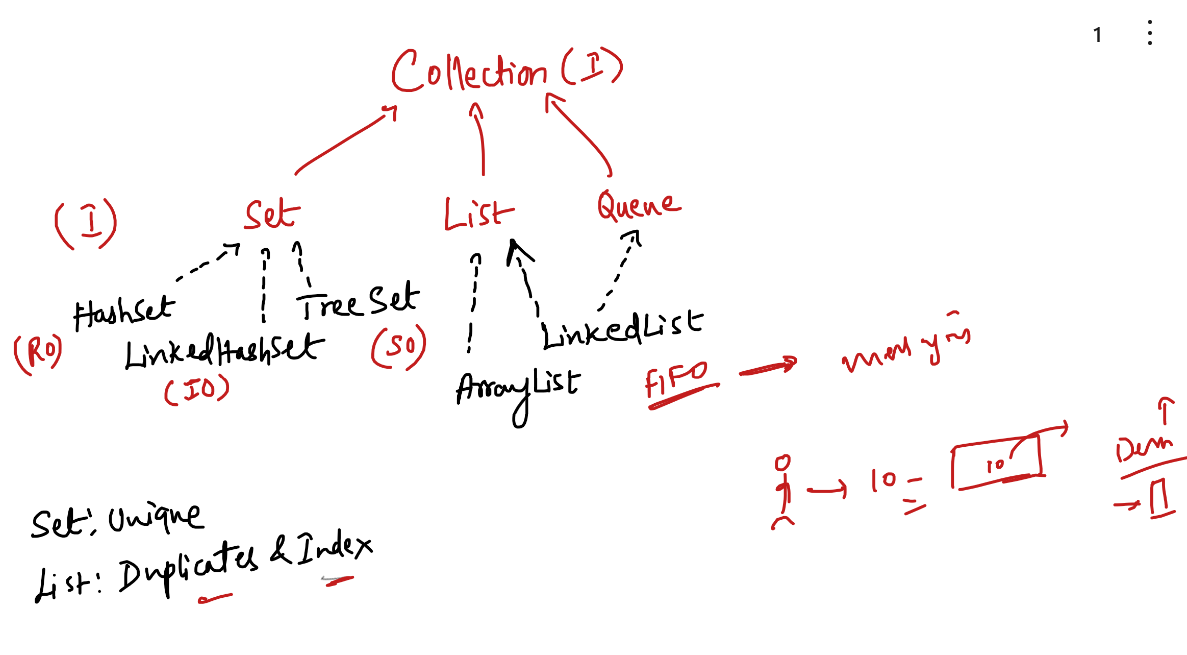
x = 20; // 30, 10, 20, 50, 40 >> sorted order 10, 20, 30, 40, 50  
y = 30;  
Integer.compare(x, y) : returns -ve  
Integer.compare(x, x) : returns 0  
Integer.compare(y, x): returns +ve  
a = 20.0  
b = 30.0  
Double.compare(a, b): returns -ve  
Double.compare(a, a): returns 0  
Double.compare(b, a): returns +ve  
  
String s1 = “hello”; // ascii code of h is higher  
String s2 = “Hello”; // ascii code of H is lower  
s1.comapreTo(s2); // returns -ve, 0 or +ve by using the ascii code of hello & Hello  
  
LocalDate dob1 = LocalDate.parse(“2000-10-25”);  
LocalDate dob2 = LocalDate.parse(“2000-11-26”);  
dob1.compareTo(dob2); // returns -ve, 0 or +ve

Summary:

1. interfaces & abstract classes
2. Exception Handling
3. Object, String, Integer, Double, LocalDate
4. compareTo, compare() : either give +ve or 0 or -ve

Collection Framework: It provides set of interfaces & classes to maintain the data & it is dynamic in nature(it grows its size at runtime & reduces its size at runtime while you add/remove data)



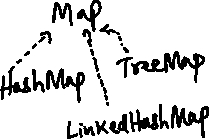
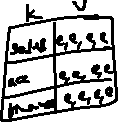
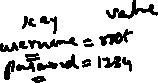


All these are inbuilt classes already available in Java, you need to use these classes as per your requirement

Common methods of these implementation are   
add(item): to add the elements  
remove(item): to remove the elements  
iterator(): to iterate over the elements  
size(): to find the number of elements  
clear(): to remove all the elements

Map: It is also dynamic in nature, it is an interface which maintains the data in key & value pairs, it is not part of collection hierarchy

Map methods:   
put(key, value): Store the key & value  
get(key): Retrieve the value from key  
remove(key): Remove the value from the key

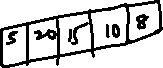


HashMap: Maintains elements in random order  
LinkedHashMap: Maintains elements in insertion order  
TreeMap: Maintains elements in sorted order  
Note: Key will be unique in Map

Iterator: It is an interface used to iterate over collection, it has 3 methods, its main purpose is to remove the elements while iterating, if you are not removing the elements then you can use for-each loop for iteration

1. hasNext: Checks whether next element exists
2. next: Refers to the next element & returns that element
3. remove: Removes the element that is iterated

List list = new LinkedList();  
list.add(5); list.add(20); list.add(15); list.add(10); list.add(8);  
list = [5, 20, 15, 10, 8]  
Iterator it = list.iterator();  
while(it.hasNext()) {  
 int x = it.next()  
 if(x == 15) it.remove();  
}



Collection can store heterogenous value as well as same type of values

List list1 = new LinkedList(); // heterogenous  
list1.add(“hello”); list1.add(35); list1.add(employee); list1.add(true);  
List<Employee> list2 = new LinkedList<>(); // generic type  
list2.add(employee); // works  
list2.add(“hello”); // compile time error

Comparable: It is used to sort, but it can provide only natural sorting technique as it is part of the object, it has a method called compareTo(Object obj), the method is   
 public abstract int compareTo(Object obj)

Comparator: It is also used to sort, it is not part of any object, it can be implemented separately to compare, it has a method called compare(Object x, Object y), the method is   
 public abstract int compare(Object x, Object y);

Note: TreeSet calls compareTo of an object by default, however you can supply Comparator implementation to it so that it calls the compare method of Comparator instead of calling compareTo of Comparable.

// calls compareTo of Comparable implemented in Employee  
Set<Employee> set = new TreeSet<>();

// calls compare of Comparator implemented outside Employee   
Set<Employee> set = new TreeSet<> (comparatorImpObject);  
comapratorImplObject will have compare(x, y) implemented.

How to implement Comparator

1. You can implement by creating separate classes to sort each property
2. You can also implement by creating anonymous class to sort each property