/ JAVA DB SCHOOL

Abstracts, Internal Classes, Exceptions

/ Leftovers

/ OOP Properties

Abstraction

- Depend on abstractions, not on concretions
- Objects only reveal internal mechanisms that are relevant for the use of other objects, hiding any unnecessary implementation code
- Very helpful to easily adapt the code over time when changing concrete implementations

Abstract Classes

- Allows to define methods without implementing them
- Classes which extend abstract classes will implement them

```
public abstract class GraphicObject {
   // declare fields
   // declare nonabstract methods
   abstract void draw();
}
```

Interfaces

- Allow using polymorphism

- Contains methods without implementations
  - Starting with Java 8, default and static methods may have implementation in the interface definition
  - In Java 9, private and private static methods were added

- A class *implements* multiple classes (allows multiple inheritance)
  - Remember that a class *extends* only one class

Cloneable, Comparable Interfaces

- Cloneable and Comparable interfaces (defined in java.lang package)

- Classes that implement Cloneable, must define *clone* method
  - Should create a copy of the provided object
  - https://docs.oracle.com/javase/7/docs/api/java/lang/Cloneable.html

- Classes that implement Comparable, must define *compareTo* method
  - Defines the “order” of the first object to the second one
  - https://docs.oracle.com/javase/8/docs/api/java/lang/Comparable.html

/ Internal Classes

Internal Classes

- Defined inside other classes. Improve code readability and maintainability
- Enhance encapsulation, hide internal classes in external classes, and do not allow other classes to access internal classes

```
class Extern {
    private int x;

    public void method1() {
        // code
    }

    class Intern {
        public void method2() {
            method1();
        }
    }
}
```

Internal Classes

- An internal class can access variables and methods of the class where it is defined (because it is a member of that class)

- A reference to an object of the external class must be used to access the internal class if the internal class is non-static

```
public static void main(String[] args) {
        Extern extern = new Extern();
        Extern.Intern intern = extern.new Intern();
        intern.method2();
}
```

Internal Classes

- An internal class can be static

```
Extern.Intern intern = new Extern.Intern();
```

- Inside the internal class, `this` is used to refer to it, while `Extern.this` is used to refer to external class (if it is named `Extern`)

Internal Classes

- An internal class can be static

```
Extern.Intern intern = new Extern.Intern();
```

- Inside the internal class, `this` is used to refer to it, while `Extern.this` is used to refer to external class (if it is named `Extern`)

Internal Classes

- An internal class can be defined inside a method
- Used when only a tiny class is need for some local operations inside the method

```
class Extern {
    private String x = "Extern";

    void method() {
        class Intern {
            public void show() {
                System.out.println("x=" + x);
            }
        }
    }
}
```

/ Exceptions

Exceptions vs Errors

- **Error**: An Error indicates serious problem that a reasonable application should not try to catch

- **Exception**: Exception indicates conditions that a reasonable application might try to catch

Exception Hierarchy

- All exception and errors types are sub classes of class Throwable
- Exception branch – used for exceptional conditions that user programs should catch (e.g.: NullPointerException)
- Error branch – used by the Java run-time system (JVM) to indicate errors having to do with the run-time environment itself (JRE) (e.g., StackOverflowError)
- Runtime Exception, Error are unchecked exceptions (they are not mandatory to be checked)

Catching an Exception

- An exception may be catched

```
int res =0;
try {
        res = divideByZero(a,b);
}
catch(NumberFormatException ex) {
        System.out.println("NumberFormatException is occured");
}
```

Throwing an Exception

- An exception may be thrown

```
static void checkAge(int age) {
    if (age < 18) {
        throw new ArithmeticException("Access denied - You must be
at least 18 years old.");
    }
    else {
        System.out.println("Access granted - You are old enough!");
    }
}

public static void main(String[] args) {
    checkAge(15); // Set age to 15 (which is below 18...)
}
```

Finally

- The finally statement lets you execute code, after try...catch, regardless of the result

```
try {
    int[] myNumbers = {1, 2, 3};
    System.out.println(myNumbers[10]);
} catch (Exception e) {
    System.out.println("Something went wrong.");
} finally {
    System.out.println("The 'try catch' is finished.");
}
```

Custom Exceptions

- Users can create their own (checked or unchecked) exceptions

```
public
IncorrectFileNameException(String
errorMessage, Throwable err) {
    super(errorMessage, err);

}
```

```
try (Scanner file = new Scanner(new
File(fileName))) {
    if (file.hasNextLine()) {
        return file.nextLine();
    }
} catch (FileNotFoundException err) {
    if (!isCorrectFileName(fileName)) {
        throw new
IncorrectFileNameException(
          "Incorrect filename : " +
fileName , err);
    }
    // ...
}
```

Custom Exceptions

- Custom unchecked exception:

```
public class IncorrectFileExtensionException
        extends RuntimeException {
    public IncorrectFileExtensionException(String errorMessage, Throwable
err) {
        super(errorMessage, err);
    }
}
```

/ Practice, practice, practice

List with Nodes

- Define a class named `Lista`, which contains an internal class `Node`. Node has the following members:

```
private int val;
private Nod next;
```

- Define an empty constructor for `Node` and one that receives a `val`

- Define inside the `Lista` class the following field:

```
private Nod start;
```

List with Nodes (cont.)

- Define the following self-explanatory constructors and methods inside `Lista`:

```
public Lista();
public printList();
public void add(int val);
public int size();
public void addAtPosition(int val, int position);
public void inspectPosition(int position);
public void removePosition(int position);
public void setAtPosition(int val, int position);
```

- Verify all methods running your list from a main method

Account

Define an *Account* class containing an `accountNo`, `amount` and `nationalId` property. Requirements:

- Create an empty constructor
- Implement method `public void deposit(int amount)`
- Define a `NotEnoughMoneyException`
- Implement method `public void withdraw(int amount)` which throws an `NotEnoughMoneyException` if there is not enough money into the account
- Define an `InvalidNationalIdException`
- Implement method `public void linkToNationalId(int nationalId)` which throws an `NotEnoughMoneyException` if there the national id is not valid
- Test the functions inside a `main` class

/ Q&A


OBILE/ACADEMY