COMPUTER PROGRAMMING
Unit 3 teaching guide
Elements of Object Oriented Programming. Inheritance and Exception Handling

Teachers of PRG
Departament de Sistemes Informàtics i Computació
Universitat Politècnica de València

January 25, 2018

1 Contents

1. OOP concepts
   - Inheritance
   - Java hierarchy of classes
   - Inheritance in the Java API documentation
2. Exception handling in Java
   - Hierarchy from `Throwable`
   - User-defined exceptions
   - Mechanism `try-catch-finally`
   - `throws` clause
   - `throw` instruction

2 Specific objectives

Upon completion of this subject the student should be able to ...

Objective 1 explain why the use of inheritance is recommended in Object Oriented Programming.

Objective 2 explain the hierarchy of Java predefined classes as they appear in the Java API documentation.

Objective 3 apply the different qualifiers for instance variables and methods when using inheritance.

Objective 4 use concepts related to inheritance as method overriding in the design of derived classes.

Objective 5 explain the advantages of using exceptions in OOP.

Objective 6 distinguish between errors (unrecoverable situations) and exceptions (recoverable situations).

Objective 7 design new Java classes using inheritance.

Objective 8 use pre-defined classes derived from class `Exception` taking into account their respective position in the hierarchy of exceptions.

Objective 9 design new exceptions as Java classes derived from pre-defined exceptions.

Objective 10 distinguish between checked and unchecked exceptions.

Objective 11 apply the `try-catch-finally` mechanism for handling exceptions or use the `throws` clause for propagating them.

Objective 12 explain how the propagation mechanism operates.

3 References

- "Empezar a programar usando Java"
  N. Prieto, A. Casanova, et al.
  Editorial UPV
  Chapters 14 and 15
- "Introduction to Programming Using Java, Sixth Edition"
  David J. Eck
  `http://math.hws.edu/javanotes/`
  Chapter 3 (3.7), Chapter 5 (5.5 & 5.6) & Chapter 8 (8.3)
- "Estructuras de datos en Java: compatible con Java 2"
  Mark Allen Weiss
  Ed. Addison Wesley, 2000 - 2006
  Chapter 2 (2.5) & Chapter 4 (4.1, 4.2 i 4.3)
- "The Java^{TM} Tutorials Oracle"
  `http://download.oracle.com/javase/tutorial/`
  Trail: Learning the Java Language. Lesson: Interfaces and Inheritance
  Trail: Essential Java Classes. Lesson: Exceptions.

4 Planning of each session

Duration of activities

	Inside classroom	Outside classroom
Before	–	2h
Session 1	1.5h	2h
Session 2	1.5h	2h
Session 3	1.5h	2h
	4.5h	8h

Before the first session

Outside classroom activities (up to 2h)

- It is proposed to read chapter 14 of "Empezar a programar usando Java".
- And to read sections 5.5 and 5.6 of "Introduction to Programming Using Java, Sixth Edition".

First session

Classroom activities (1h 30')

Minutes	Activity
20'	The teacher explains. Students will have available a copy of the slides to take notes. Introduction to the concept of inheritance. Examples of inheritance.
5'	Break.
20'	Relation of inheritance. Hierarchy of classes. Inheritance in the Java API Documentation.
15'	Methods overriding. Examples.
20'	Discussion: Life classification as an example of hierarchical organisation. Let's suppose you are a member of the development team of the *"Museo de las Ciencias Principe Felipe"*, and the museum manager ask your team for developing a computer program for representing all kinds of known life. The application will be user friendly since kids will interact with it. Here we are going to do part of the initial analysis.
10'	Resolution of doubts.

Outside classroom activities (up to 2h)

- Review of basic concepts explained in class by reading sections of books proposed below. Concepts to be reviewed: inheritance, subclass (or derived class), superclass, `this` and `super`.
  - Chapter 14 of "Empezar a programar usando Java".
  - Sections 5.5 and 5.6 of "Introduction to Programming Using Java, Sixth Edition".
- Study of new concepts such as: exception handling, `try-catch-finally` mechanism, `throws` clause and `throw` instruction. See some of Java predefined classes for exception handling.
  - Chapter 15 of "Empezar a programar usando Java".
  - Sections 3.7 and 8.3 of "Introduction to Programming Using Java, Sixth Edition".

Second session

Classroom activities (1h 30')

Minutes	Activity
20'	The teacher explains. Introduction to exception handling. User-defined exceptions.
5'	Break.
20'	What can we do once an exception has been thrown? Checked and unchecked exceptions. `try-catch-finally`
5'	Break.
15'	What can we do once an exception has been thrown? Example of exception handling using propagation and catching.
15'	Teacher explains the details of a complete example. See classes `AskingForDate`, `IllegalDateException` and `TestAskingForDate`.
10'	Resolution of doubts.

AskingForDate.java

```
import java.util.*;
import java.io.*;

/**
 * Example of class for asking the user for the three components a date.
 * Objects of this class will be used reading values from standard input
 * and checking if the complete date is correct.
 *
 * @author Jon Ander Gómez (jon@dsic.upv.es)
 * @version 1.0 (March 2012)
 */
public class AskingForDate
{
    /**
     * Class variable for maintaining the number of days per month in a year.
     */
    private static int daysPerMonth[] = { 0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };

    /**
     * Instance variable for storing the reference to the object of the class
     * Scanner related to standard input.
     */
    private Scanner input;

    /**
     * Constructor for preparing objects of this class.
     *
     * @param input Object of the Scanner class.
     */
    public AskingForDate( Scanner input )
    {
        if ( input != null )
            this.input = input;
        else
            this.input = new Scanner( System.in );
    }

    /**
     * Method for reading a date from standard input and checking if it is a legal date.
     *
     * @param prompt <b>String</b> object for prompting to the user.
     *
     * @return An object of class <b>Date</b> representing the date typed by the user.
     */
    public Date readDate( String prompt )
    {
        boolean dateOK;
        int     dayOfMonth=0, month=0, year=0;

        do {
            dateOK=true;

            System.out.printf( "%s (dd mm yyyy): ", prompt );

            try {
                dayOfMonth = readInt( 1, 31 );
                month      = readInt( 1, 12 );
                year       = readInt( 1, 3000 );

                checkDate( dayOfMonth, month, year );
```

```
            }
            catch( InputMismatchException ime )
            {
                dateOK=false;
                System.out.printf( "ERROR: %s\n", ime.getMessage() );
                input.nextLine(); // Clean input buffer
            }
            catch( IllegalDateException ide )
            {
                dateOK=false;
                System.out.printf( "%s\n", ide.getMessage() );
                input.nextLine(); // Clean input buffer
            }
        } while( !dateOK );


        Calendar cal = Calendar.getInstance();
        cal.set( year, month-1, dayOfMonth );

        return cal.getTime();
    }

    /**
     * Method for reading an integer value from standard input and checking if it is
     * in the specified range.
     *
     * @param lowerBound Integer value representing the minimum accepted value.
     * @param upperBound Integer value representing the maximum accepted value.
     *
     * @return The integer value type by the user.
     */
    private int readInt( int lowerBound, int upperBound )
        throws InputMismatchException
    {
        int value = input.nextInt();

        if ( value < lowerBound || value > upperBound ) {
            String errorMsg = String.format( "%d is not in [%d,%d]",
                                             value, lowerBound, upperBound );
            throw new InputMismatchException( errorMsg );
        }

        return value;
    }

    /**
     * Private method for checking if a given date is a legal date.
     * This method throws an exception of the user-defined class <b>IllegalDateException</b>.
     *
     * @param d  Day of month.
     * @param m  Month of the year in the range [1,12].
     * @param y  Year.
     */
    private void checkDate( int d, int m, int y )
        throws IllegalDateException
    {
        if ( m < 1 || m > 12 )
            throw new IllegalDateException( "Incorrect value for the month!" );

        boolean leapYear = ( (y%4)==0 && (y%100)!=0 ) || (y%400)==0;
```

```
        int maxD = daysPerMonth[m];
        if ( leapYear && m == 2 ) maxD++;

        if ( d < 1 || d > maxD )
            throw new IllegalDateException( "Incorrect value for the day of the month!" );
    }
}
```

AskingForDate.java

IllegalDateException.java

```
/**
 * User-defined exception.
 *
 * @author Jon Ander Gómez (jon@dsic.upv.es)
 * @version 1.0 (March 2012)
 */
public class IllegalDateException
    extends Exception
{
    /**
     * Constructor for preparing objects of this class.
     *
     * @param msg Error message describing the conditions that cause this exception.
     */
    public IllegalDateException( String msg )
    {
        // Call to one constructor of the superclass.
        super( "Illegal date: " + msg );
    }
}
```

IllegalDateException.java

TestAskingForDate.java

```
import java.util.*;

/**
 * Class for testing the use of objects of the class <b>AskingForDate</b>.
 *
 * @author Jon Ander Gómez (jon@dsic.upv.es)
 * @version 1.0 (March 2012)
 */
public class TestAskingForDate
{
    /** <em>friendly</em> class variable for reading data from standard input */
    static Scanner input = new Scanner( System.in );

    /**
     * <code>main()</code> method for testing the use of objects of other classes.
     *
     * @param args Array of objects of class <b>String</b> with the list of
     *             command line arguments.
     */
    public static void main( String args[] )
    {
        AskingForDate afd = new AskingForDate( input );

        Date d = afd.readDate( "Birth date" );

        System.out.println( "You were born " + d.toString() );
    }
}
```

TestAskingForDate.java

Outside classroom activities (up to 2h)

- Given the file `Life.jar` available in *PoliformaT*, which includes the tree representation of a subset of living beings, **workgroups must complete the existing classes (the ones provided in `Life.jar`) with the corresponding attributes (instance variables), common attributes (static variables or class variables) and methods.** Both types of attributes where enumerated as features in the discussion at the end of the first session. All attributes must be private, so at least one getter method for each attribute must be implemented as public, and at least one setter method for each modifiable attribute.

  Inheritance rules must be applied properly: method overriding, use of `super` when required, etc.
- Constructor and `toString()` method must be written for each class.
- A `JAR` file using the same schema of `Life.jar` must be generated and presented as **Deliverable #3.1**

  See details of how to prepare the `JAR` file in the `Makefile` contained in `Life.jar`

  For extracting the contents of `Life.jar` execute: `jar xvf Life.jar`
- This deliverable must be send by e-mail before the beginning of the third session.

Third session

Classroom activities (1h 30')

Minutes	Activity
30'	Teacher explains a possible solution for Deliverable #3.1 Students contribute to the collective solution with their comments and proposals.
5'	Break.
30'	Using the `try-catch-finally` mechanism for avoiding that our programs stop their execution abnormally, for example when an `InputMismatchException` is thrown while they interact with the user. Students should write a Java class with one method for reading values of the following data types from standard input: `byte`, `short`, `int`, `long`, `float`, `double` and `String`. An object of class `Scanner` must be used for reading from standard input. Teacher helps students during the development of this exercise.
5'	Break.
10'	Teacher shows and explains a possible solution of the previous problem.
10'	Resolution of doubts.

Outside classroom activities

- No activities are planned after the last session of this topic, read the teaching guide for the next unit, where you will find activities planned to be performed before the first session.

5 Issues and problems that, minimally, should be studied in this topic

- Using predefined classes that are derived classes from other predefined ones and make use of the properties related with inheritance.
- Design classes as derived from others with different combinations of the qualifiers for methods and attributes. The goal is that students understand the proper use of these qualifiers depending on the requirements of the program being implemented.
- Analysis and study of the hierarchic relations between Java predefined classes and how this information is presented in the Java API Documentation.
- Design of little programs for catching predefined exceptions.
- Design of little programs for propagating predefined exceptions.

6 Deliverables

Id	Description	Modality	Submission date	Points
#3.1	Development of the Java classes provided in the JAR file.	In group	Before the beginning of session 3.	40

7 Qualification

The grade for all activities performed in this topic will be computed as the sum of the points of each deliverable. The grade of activities performed in group will be the same for each member of the workgroup.

The knowledge of this subject will be assessed in the quizzes and in the lab practises, the grade obtained in the deliverables described herein is used to measure the work of formation, and is part of the NAS (Grade of the Follow-up Activities, the Spanish acronym is used), which, in turn, as said in the evaluation rules, contributes to the final score with a 20%.