Object Oriented Programming

PART II

Exceptions, Tables

Beniamin Bogosel

Aurel Vlaicu University, Arad

Error handling

⋆ **Why?** we should not expect the user to behave correctly
⋆ In case unexpected behavior happens: the program should not crash
⋆ When using OOP, we expect the user to try to interact with the program in a natural way: we must be able to recover from errors

Example

```
public class X {
    private int[] tab;
    public X() {
        tab = new int[6];
    }
    public int getElem(int i) {
        return tab[i];
    }
    public static void main(String[] args) {
        X x = new X();
        int n = new Scanner(System.in).nextInt();
        System.out.println(x.getElem(n));
        System.out.println(n * n);
    }
}
```

Try various inputs to observe the results.

Execution flow in Java

⋆ execute the main function
⋆ if some other function is called, run the block of instructions
⋆ do this recursively until everything is executed

⋆ If an error occurs anywhere in the program, **it propogates** and stops the initial program.
⋆ **Exception**: object which signals a unusual situation during the execution of a program

- evaluating an expression
- charging a class or an interface
- going beyond the limits of a ressource
- executing a `throw` instruction
- internal error

How to handle exceptions?

⋆ when an exception occurs, the problem must be handled explicitly and sorted out
⋆ **error detection code**: part of the code where we watch for errors
⋆ in case an error occurs, we interrupt the code and we don't execute the instruction following afterwards
⋆ **zone handling the problem**: alternative code that will be run in case an error occurs

Re-doing previous example

```
public class X {
    private int[] tab;
    public X() {
        tab = new int[6];
    }
    public int getElem(int i) {
        return tab[i];
    }
    public static void main(String[] args) {
        X x = new X();
        int n = new Scanner(System.in).nextInt();
        try {
            System.out.println(x.getElem(n));
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Valeur trop grande");
        }
        System.out.println(n * n);
    }
}
```

Try-catch-finally

```
try {
    // zone treating a problem
} catch (Exception1 me1) {
    // zone treating exceptions of type Exception1
} catch (Exception2 me2) {
    // zone treating exceptions of type Exception2
} catch...
} finally {
    // zone treating the other cases
}
```

Exception handling

- `int a = 10/0;` generates an Exception Object which is "thrown"
- if this exception object is handled then the rest of the program is executed
- otherwise the program is stopped and the description of the exception is shown

Exception reasons

- invalid user input
- device failure
- loss of network connection
- physical limitations (out of memory)
- code errors
- out of bound
- null reference
- type mismatch
- open an unavailable file
- database errors
- arithmetic errors

Errors: irrecoverable conditions, beyond the control of the programmer

Built-in exceptions

- `ClassNotFoundException`: trying to load a class which is not found or missing
- `FileNotFoundException`: thrown when the program tries to open a file that doesn't exist
- `ArithmeticException`: thrown when an exceptional arithmetic condition has occured
- `ClassCastException`: trying to cast an object to a class it does not belong to
- `ArrayIndexOutOfBoundsException`: occurs when we try to access an element of an array with invalid index
- user defined exceptions

Handling exceptions

```
try {
    // Code that may throw an exception
} catch (ArithmeticException e) {
    // Code to handle the exception
} catch(ArrayIndexOutOfBoundsException e){
    //Code to handle the anothert exception
}catch(NumberFormatException e){
    //Code to handle the anothert exception
}
```

Arrays

```
char[] t = new char[] {'s', 'a', 'l', 'u', 't'};
```

⋆ the domain of variation for the indices is `0,..,t.length-1`
⋆ an Instance of an Array is **not a dynamical structure**

- the size of a array is `final int length`
- it is defined when the array is created
- the size of an array entity is unknown when it is declared: `int[] x;`
- It is known once the array is **instantiated**

Simple arrays of primitive types

⋆ Create an array in three steps:

- Declaration: `int[] x;`
- construction: `x = new int[2];`
- Initialization: `x[0] = 1; x[1] = 2;`

⋆ all at once (the compiler determines the size)

```
int[] x = new int[] {1, 2};
```

Simple array of objects

⋆ Create an array in three steps:

- Declaration: `Box[] b;`
- construction: `b = new Box[2];`
- Initialization: `b[0] = new Box("X",0); b[1] = new Box("Y",0);`

⋆ all at once (the compiler determines the size)

```
Box[] b = new Box [] {
    new Box("X",0), new Box("Y",0)
};
```

Multi-dimensional arrays

⋆ one dimensional array: simple array

⋆ n dimensional array = simple array of $n-1$ dimensional arrays

```
int[][] t = new int[][] {{1,2,3}, {4,5}};
```

- `t` is of type `int[2][]`
- `t[0]` is of type `int[3]`
- `t[1]` is of type `int[2]`

Resize-able arrays: `ArrayList`

- `import java.util.ArrayList;` – imports the ArrayList class
- `ArrayList<String> cars = new ArrayList<String>();` – creates a resizeable array of strings
- type `.` to see all methods avaible
- `cars.add("Dacia");` adds a string to the array
- `cars.add(0,"Mazda");` adds an element on the first position of an array
- Access element `cars.get(0);`
- Change element `cars.set(0,"Opel");`
- Remove an element `cars.remove(0);`
- Find the size `cars.size();`
- Same operation for all `cars.forEach(System.out::println);`
- iterator: `for(String s : cars){ ... }`

Test Case: student management

- ⋆ Create a `Student` class with some relevant fields
  - id as an integer
  - name as string
  - grade as integer
- ⋆ create a constructor
- ⋆ introduce methods to **get** and **set** all attributes
- ⋆ Create a student management system
  - store stdents in `ArrayList`
  - method `addStudent` adds new students
  - method `viewStudents` shows all students
  - method `updateStudent` changes attributes for a student
  - method `deleteStudent` deletes a student with given id
  - method `searchStudentByName` searches student given its name