PGR112 – Step 11: More stuff + large task 2

Object Oriented Programming

Bogdan Marculescu / Bogdan.Marculescu@kristiania.no

Agenda

- More exception handling
- The life cycle of an object
- High cohesion, low coupling
- Pass by value / reference
- Some small picks
- About the week's assignment

Exceptions

We have seen this before:

- What they are.
- How they occur.
- How they are passed on in the call stack
- The difference between checked and unchecked exceptions
- Catch or specify
- Try / catch / finally

Today

- How to cast your own exceptions?
- Multi-catch
- Try-with-resources
- Chained exceptions

Throwing your own Exceptions

- We can actively choose to throw an exception (such as in exercise 5 in step 10):

```
/*
I think addPerson is more readable if I delegate the validation to
a separate helper method like this. Agree?
 */
private void validatePerson(Person p) {
    if (p == null) {
        throw new IllegalArgumentException("Null input when Person excepted");
    }
    if (p.getAge() < 0) {
        throw new IllegalArgumentException("Please ensure valid age (>0) when adding person");
    }
    if (p.getName() == null) {
        throw new IllegalArgumentException("Please ensure valid name (not null) when adding person");
    }
}
```

Your own Exception types

- If we do not find an appropriately available exception in the Java library, then we can create our own. Then you should :
  - Find out which exception you want to extend (i.e. inherit from): Checked or Unchecked?
  - Create constructors which harmonize with the constructors in the super class.
  - Be sure to call your class <descriptive name> Exception

Example

- Example from codejava.net

```java
public class StudentNotFoundException extends Exception {

    public StudentNotFoundException(String message) {
        super(message);
    }
}
```

```java
public class StudentStoreException extends Exception {

    public StudentStoreException(String message, Throwable cause) {
        super(message, cause);
    }
}
```

Chained exceptions

- When an exception is discarded in connection with the capture of another exception. Example from Oracle:

```
try {

} catch (IOException e) {
    throw new SampleException("Other IOException", e);
}
```

```
1  public void save(Student student) throws StudentStoreException {
2      try {
3          // execute SQL statements..
4      } catch (SQLException ex) {
5          throw new StudentStoreException("Failed to save student", ex);
6      }
7  }
```

Multi-catch

- Appropriate if you have many different types of exceptions you catch and want to treat everyone the same way. Example from Oracle (pre Java 7):

```
catch (IOException ex) {
     logger.log(ex);
     throw ex;
catch (SQLException ex) {
     logger.log(ex);
     throw ex;
}
```

- Post Java 7:

```
catch (IOException|SQLException ex) {
    logger.log(ex);
    throw ex;
}
```

Try-with-resources

- Example from baeldung (pre Java 7):

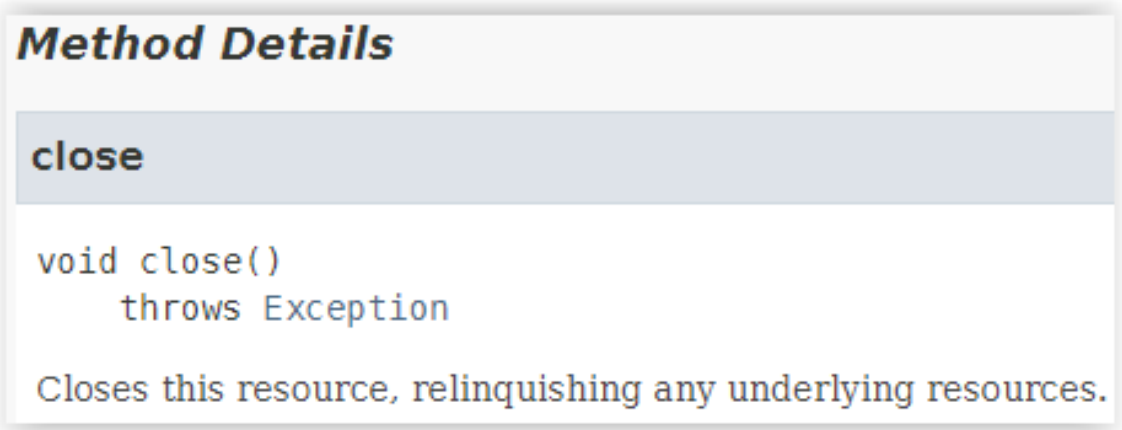
```
Scanner scanner = null;
try {
    scanner = new Scanner(new File("test.txt"));
    while (scanner.hasNext()) {
        System.out.println(scanner.nextLine());
    }
} catch (FileNotFoundException e) {
    e.printStackTrace();
} finally {
    if (scanner != null) {
        scanner.close();
    }
}
```

- Post Java 7:

```
try (Scanner scanner = new Scanner(new File("test.txt"))) {
    while (scanner.hasNext()) {
        System.out.println(scanner.nextLine());
    }
} catch (FileNotFoundException fnfe) {
    fnfe.printStackTrace();
}
```

Try-with-resources

- We do not need to close the resource explicitly (for example in a finally).
- The resource must implement the AutoCloseable interface:



Method Details

close

```
void close()
    throws Exception
```

Closes this resource, relinquishing any underlying resources.

The life cycle of an object

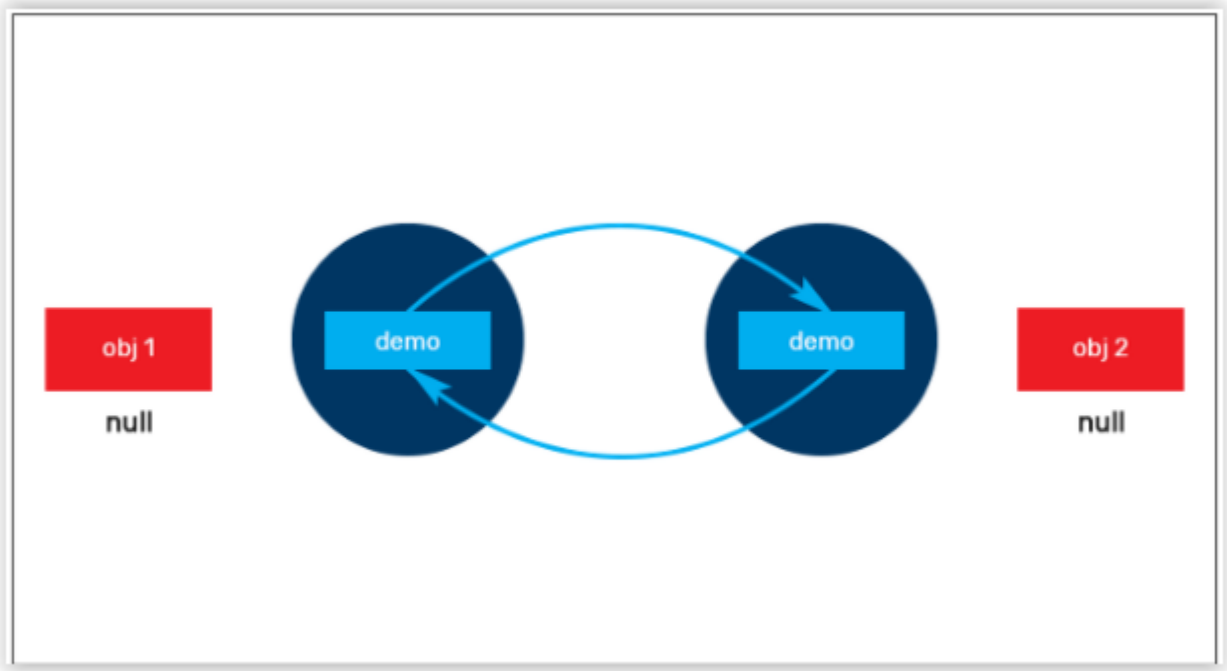
- We know how an object is “born” (well, created, anyway)
  - Person p = new Person();
- What do objects “die”?
  - When no one refers to the object anymore. – I.e. there is no reference to the current object.
  - Garbage collector frees the memory.

Cleanup

- The book explains this in a nice way (see Chapter 20: Garbage Collection)
  - Nullifying the reference variable
  - Re-assigning the reference variable
  - Creating an object inside a method
  - Island of isolation
- In Java, we have a Garbage Collector that cleans up objects (removes from memory) when they are no longer needed (because they are not referenced).

A little demo of life cycles

- Covering 3 of 4 cases in the book (could be worse…)



[medium]

High cohesion – low coupling

- Cohesion: the degree to which a part of a code base forms a logically single, atomic unit.
  - A class should have only one area of responsibility (Single responsibility principle)
  - A method should perform "one thing", not "many things"
  - Encapsulation
- Coupling: the degree to which a single unit is independent from others.
  - Inheritance vs aggregation
  - Dependence on classes (implementations) vs interface (abstraction)

- Ideally, we want object oriented code we write to have *high cohesion* and *low coupling*.
- You can read more here.

Pass by value / reference

- (See the book, Chapter 19)
- Deals with how the input we send to a method behaves in relation to whether they are primitive data types or object references.
  - Primitive data types: *the value* gets sent to the method.
  - Object references: *the reference* is sent to the method (technically it's a value of a reference, but you know)
- What we send to a method is called an argument (one or more)
- What the method says it can accept is called parameters (one or more)

Demo – pass by value/reference

- Demo time – hopefully will clarify things…

Conditional operator ? :

```
public static void main(String args[]) {
   int a, b;
   a = 10;
   b = (a == 1) ? 20: 30;
   System.out.println( "Value of b is : " +  b );

   b = (a == 10) ? 20: 30;
   System.out.println( "Value of b is : " + b );
}
```

Output:

Value of b is : 30

Value of b is : 20

var

- From Java 10.
- We can use var instead of specifying the type when we declare a variable
  - For local variables (in methods)
  - Only if we instantiate in the same line
- Example (from codejava):
  - Pre Java 10:

```
1 | List<String> list = Arrays.asList("One", "Two", "Three", "Four", "Five");
```

  - Post Java 10:

```
1 | var list = Arrays.asList("One", "Two", "Three", "Four", "Five");
```

- The compiler understands what type of variable to have (provided immediate initialization).

About the task this time

- Choose whether you want to start from the starting point in the github repo, or with your own code.
- Expand your solution little by little. Test that each small extension works along the way.
- Feel free to collaborate with others!
- No crisis if not all functionality is in place. As long as some functionality is in place, you have come a long way! Ever better if the functionality that is there works well and is tested.
- Feel free to use peer review!
- We will discuss the topic later in the course.
- Therefore a good idea to collaborate with others and peer review.
- As the assignment is large, you will have some time to work with it. You can continue as we move on with the course.

Some things to think about

- How do I make sure that all the characters can be printed? Think «å» or other special characters.
- Where do I start?
  - I recommend starting by reading the contents of the file. That is the biggest part of the task. You have come a long way when you manage to read all the different types of objects.
  - Does it sound like that responsibility can all be in one class?
- **OBS!** Worth remembering:
  - Reading and writing double values. In order for Scanner to be able to read double values from the file there «.» is used to distinguish between whole value and partial value (ex 5.6), then it must use locale: Locale.US

```
try(Scanner scanner = new Scanner(new File(fileName))){
    scanner.useLocale(Locale.US);
```

Conclusion

- Goals for this step:
  - I am familiar with extended exception handling such as multi-catch, exception-chaining and try-with-resources.
  - I know the life cycle of an object and how the Garbage Collector cleans up objects.
  - I understand the concepts of high cohesion and low coupling.
  - I understand what is meant by the terms pass by value and pass by reference.

Good luck with the larger task.