Exception Handling

Name: _____ ID: _____

These questions do not have a formal, definitive answer. They are meant to be food for thoughts. Feel free to seek answers on browsing the Internet, talking to other software developers or reading books.

1. There are many different ways to handle exceptional conditions in programming languages. Below I am enumerating five of them. For each method, describe its advantages and disadvantages.

   (a) Use guards to avoid going into exceptional conditions:

   ```
   public int pop() {
     Node n = top;
     top = n.getLink();
     return n.getData();
   }
   ...
   f (s.hasMore())
     s.pop();
   ```

   (b) Use special values to denote error situations:

   ```
   public int pop() {
     Node n = top;
     if (n==null) return 0;
     top = n.getLink();
     return n.getData();^K
   }
   ```

   (c) Stop everything and break the world into many pieces:

   ```
   public int pop() {
     Node n = top;
     if (n==null) {
       System.out.println("Popping an empty stack!");
       System.exit(-1);
     }
     top = n.getLink();
     return n.getData();
   }
   ```

   (d) Use a global variable to flag errors. This is the approach of the "old-school" C programmers:

```
#include <stdio.h>
#include <errno.h>
void checkArgs (int argc, char** argv) {
  if (argc < 2) {
    errno = 1;
  }
}
int main(int argc, char** argv) {
  checkArgs(argc, argv);
  if (errno == 1) {
    fprintf(stderr, "Syntaxe: %s file\n", argv[0]);
  } else {
    printf("Argument = %s\n", argv[1]);
  }
}
```

(e) Use a language that allows you to define pre and post-conditions, like *Eiffel*:

```
balance: INTEGER -- Current balance
deposit (sum: INTEGER)
  require
    non_negative: sum >= 0
  do
  balance := balance + sum
  ensure
    updated: balance = old balance + sum
  end
```

2. In the next three questions you must tell what will happen in each execution of the following program:

```
public class TestException {
  public static void main(String[] args) {
    int i = Integer.parseInt(args[0]);
    int j = Integer.parseInt(args[1]);
    System.out.println(i / j);
  }
}
```

(a) `java TestException 6 3`

(b) `java TestException 6`

(c) `java TestException 6 0`

(d) `java TestException 6 asdfg`

3. Java uses "exceptions" to handle exceptional conditions. What is an exception, after all? I mean, what is this way to handle errors? Is it the same as one of the ways seen before?

4. Java represents exceptions through an hierarchy of classes: `Throwable`, `Error`, `Exception` and `RuntimeException`. What is the purpose of each of these classes?

5. Exception, at least as handled in Java, is a kind of a *design pattern* implemented at the language level. This pattern consists on a number of constructions. Explain each of these constructions:

   (a) The declaration of the exception object.

   (b) The throwing of the exception

(c) The catching of the exception.

6. Modify the program seen in exercise 2, so that it will catch and handle each of the three main exceptions that may be triggered during its execution.

7. How to use the object hierarchy to minimize the group the catching codes into the same block?

8. What happen after we finish handling the exception in the catch block? That, where the program flow goes, after we leave a catch block?

9. What would happen in the program below?

```
public class TE4 {
  public static void main(String[] args) {
    try {
      int i = Integer.parseInt(args[0]);
      int j = Integer.parseInt(args[1]);
      System.out.println(i / j);
    } catch (Exception e) {
      System.out.println("You are out of the array!");
      e.printStackTrace();
    } catch (ArithmeticException ae) {
```

```
        System.out.println("You are dividing by zero!");
        ae.printStackTrace();
      }
    }
  }
```

10. How can I declare an exception in Java? Illustrate this with an example.

11. Java has the concept of *checked* and *unchecked* exceptions. What is the difference between each type? Which exceptions belong into a category, and which exceptions belong into the other?

12. What are the advantages and disadvantages of checked exceptions?

13. A method that can get a checked exception cannot ignore it. It can handle the exception, with a try/catch block, or it can pass the exception forward. How does a method pass an exception forward?

14. What does this program do? What if I uncomment line 7, and remove line 6?

```
1  public class MT1 {
2   public static void main(String args[]) {
3     System.out.println("1");
4     try {
5       System.out.println("2");
6       if (true) throw new Exception();
7       // if (true) throw new Error();
8       System.out.println("3");
9     } catch (Exception e) {
10      System.out.println("4");
11    } finally {
12      System.out.println("5");
13    }
14    System.out.println("6");
15  }
16}
```

15. What about this program?

```
public class MT2 {
  public static int f(boolean t) {
    try {
      if (t) throw new Exception();
      return 1;
    } catch (Exception e) {
      System.out.println("2");
    } finally {
      System.out.println("3");
    }
    System.out.println("4");
    return 2;
  }
  public static void main(String args[]) {
    System.out.println("1");
    f(false);
  }
}
```