innopolis university

Exceptions and testing

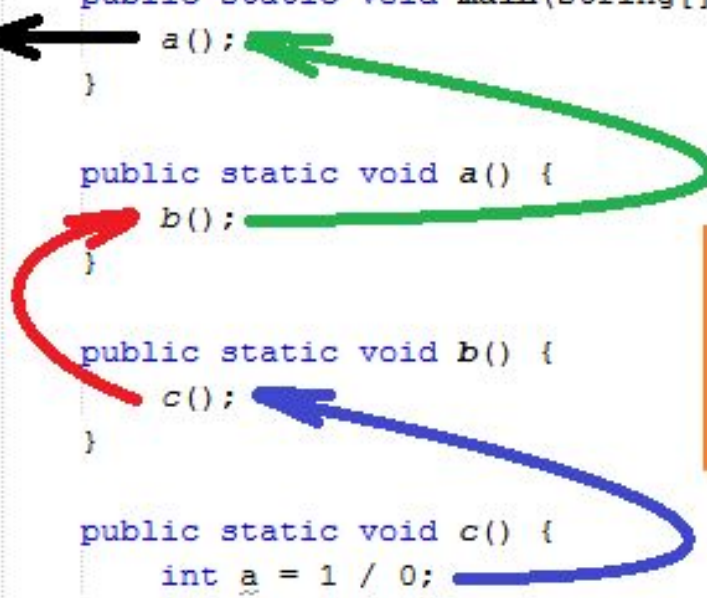
10/08/2016

Problems (errors) vs Exceptions

- **"Problem" – situation when your program behaves not in expected way**
  - `2 + 2 = 5`
  - `2 + 2 = 4` … but takes 20 seconds to calculate
  - `1 / 0 … 1.0 / 0.0`
  - `File.open("???123321\5431`");`
  - ```
    a = null;
    a.equals(b);
    ```
- **Exception – problem, that can be detected as something "not expected to happen" ("exceptional") and handled in your code. In Java language exceptions are implemented as specific kind of objects and operators to handle them.**
    ```

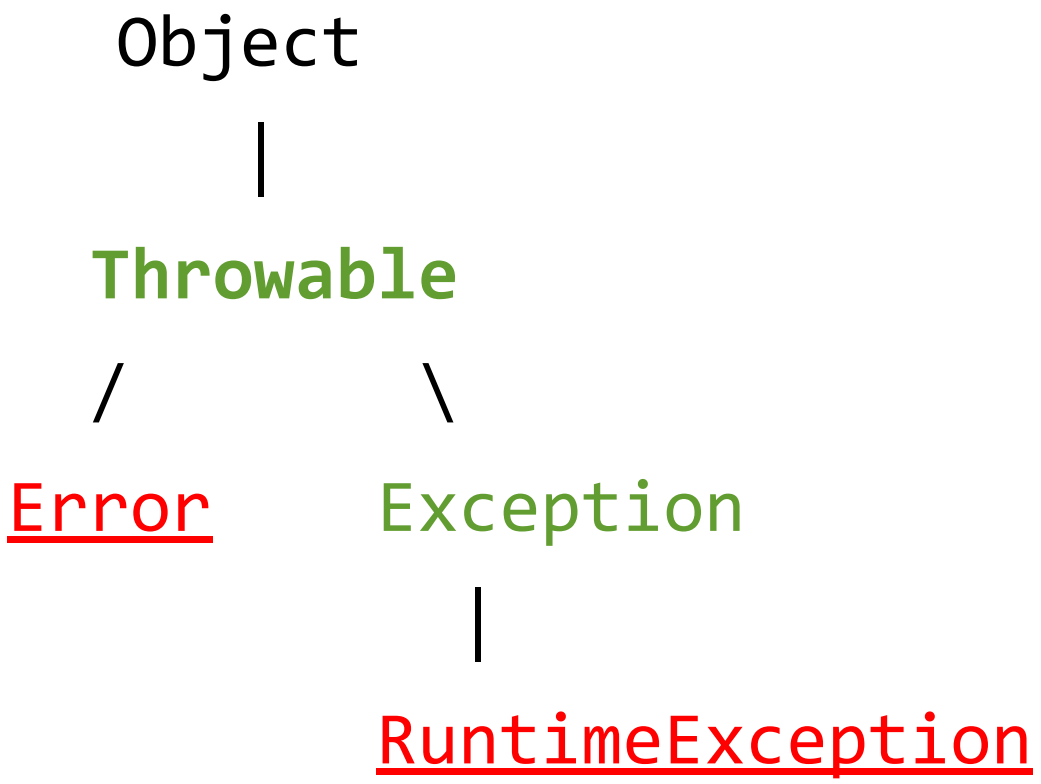
Exception nature

```
11  public class Deeper {
12
13      public static void main(String[] args) {
14          a();
15      }
16
17      public static void a() {
18          b();
19      }
20
21      public static void b() {
22          c();
23      }
24
25      public static void c() {
26          int a = 1 / 0;
27      }
28  }
```



```
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at Deeper.c(Deeper.java:26)
        at Deeper.b(Deeper.java:22)
        at Deeper.a(Deeper.java:18)
        at Deeper.main(Deeper.java:14)
```

Java Exception hierarchy

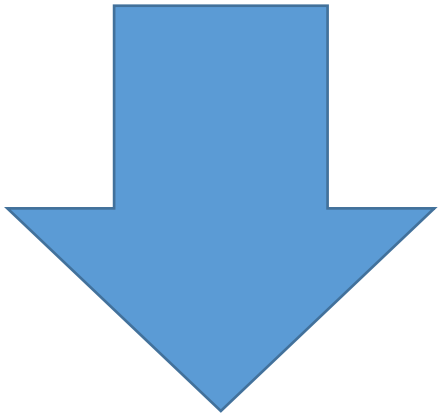


Java Exception paradigm

- Classifications
  - Compilation errors vs. runtime exceptions
  - Errors vs Exception
    - Errors are either compilation errors or serious problems that should not be handled
  - Checked ("predictable") vs Unchecked exceptions
    - All Errors are Unchecked exceptions

Java compile-time errors

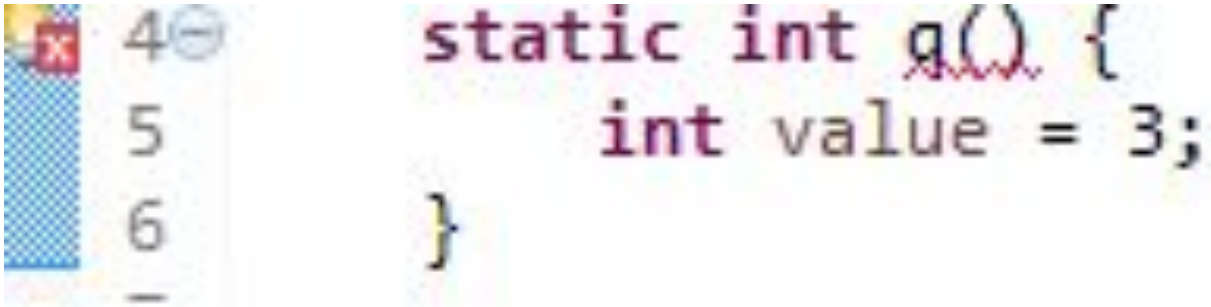
```
6        int variable = new Object();
```



```
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
        Type mismatch: cannot convert from Object to int

        at Orders.q(Orders.java:6)
        at Orders.main(Orders.java:11)
```

Java compile-time errors



```
4  static int q() {
5      int value = 3;
6  }
```

```
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
        This method must return a result of type int

        at Orders.q(Orders.java:4)
        at Orders.main(Orders.java:9)
```

Java runtime-time exceptions

```
static int usual() { return 1; }
static int fun() { while (true); }

static int q() { throw new ArithmeticException(); }
static int qGuarded() { return q(); }

static int p() throws Exception { throw new Exception(); }

static int pGuarded() {
    int val = 0;
    try {
        val = p();
        System.out.println("Everything is ok!");
        return val;
    } catch (Exception ex) {
        System.out.println("Ooops!");
        return 0;
    } finally {
        System.out.println("In any case ");
    }
}

public static void main(String[] args) {
    pGuarded();
    qGuarded();
}
```

Unchecked exception

Checked exception

Check

Try block

Catch blocks

Finally block

```
Exception in thread "main" Ooops!
In any case
java.lang.ArithmeticException
        at Orders.q(Orders.java:6)
        at Orders.qGuarded(Orders.java:7)
        at Orders.main(Orders.java:27)
```

Try – catch – finally

```
try {
    // ...
} catch (Exception e) {
    // A1: only if extends Exception
} catch (Throwable e) {
    // A2: cannot switch with A1!
} finally {
    // F : executed after catch
}
```

Try – catch – finally: special cases

```
try {

} finally {

}
```

```
static int pGuarded() throws Exception {
    try {
        throw new Exception("A");
    } catch (Exception e) {
        throw new Exception("B");
    } finally {
        throw new Exception("C");
    }
}
```

Checked Exception

What is Checked Exception in Java Programming language? In simple language: Exception which are checked at Compile time called Checked Exception. Some these are mentioned below. If in your code if some of method throws a checked exception, then the method must either handle the exception or it must specify the exception using throws keyword.

1. IOException
2. SQLException
3. DataAccessException
4. ClassNotFoundException
5. InvocationTargetException
6. MalformedURLException

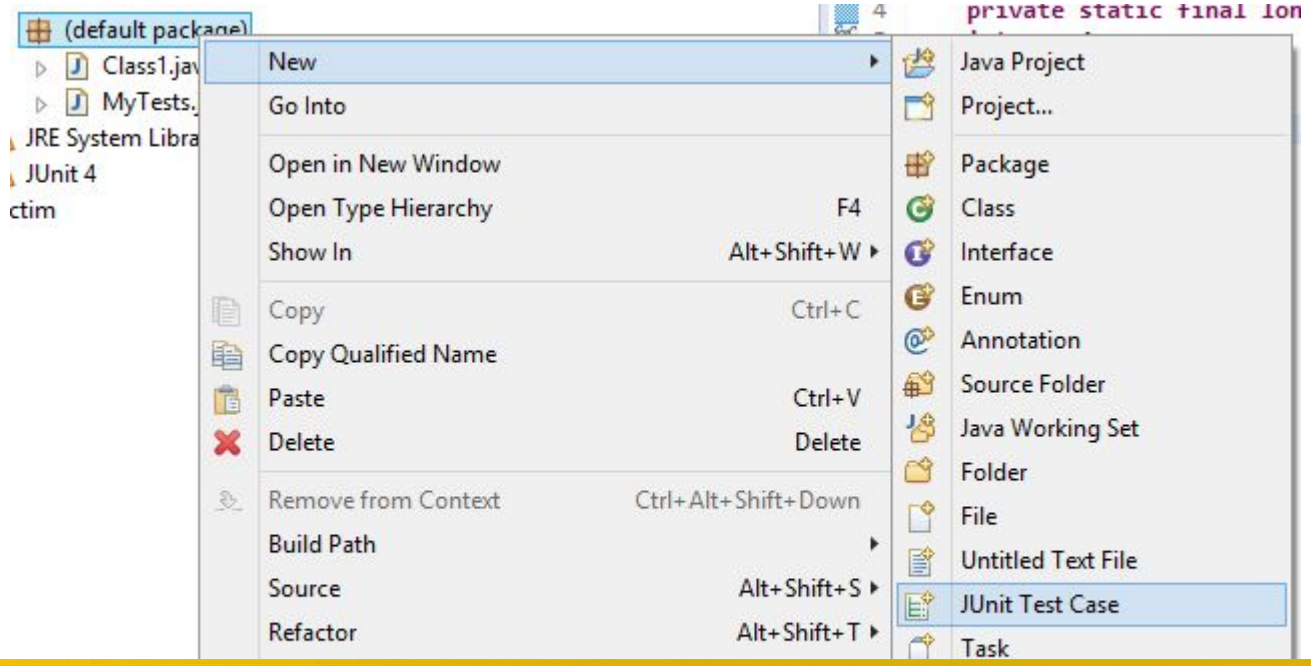
Unchecked Exception

Unchecked Exception in Java is those Exceptions whose handling is NOT verified during Compile time. These exceptions occurs because of bad programming. The program won't give a compilation error. All Unchecked exceptions are direct sub classes of RuntimeException class.

1. NullPointerException
2. ArrayIndexOutOfBound
3. IllegalArgumentException
4. IllegalStateException

Unit testing

- **Unit testing** – an approach in programming, that allows to check if parts (units) of your program behaves right in automatic way
- There are few frameworks to implement Unit test. *JUnit* is pre-installed for Eclipse.



Multiple exception handling

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

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

Rethrowing exceptions

```
catch(Exception e) {
  System.err.println("An exception was thrown");
  throw e;
}
```

```
try {
   // code here that throws an Exception
}
catch (Throwable t) {
   throw t; // (re)throw it in their face
}
```

Rethrowing exceptions

```
public static void main(String[] args) {
    try{
        rethrow("abc");
    }catch(FirstException | SecondException | ThirdException e){
        //below assignment will throw compile time exception since e is final
        //e = new Exception();
        System.out.println(e.getMessage());
    }
}

static void rethrow(String s) throws FirstException, SecondException,
        ThirdException {
    try {
        if (s.equals("First"))
            throw new FirstException("First");
        else if (s.equals("Second"))
            throw new SecondException("Second");
        else
            throw new ThirdException("Third");
    } catch (Exception e) {
        //below assignment disables the improved rethrow exception type checking
        // e=new ThirdException();
        throw e;
    }
}
```

Task 1

Implement your own exception class to handle equation solving problems. Write methods for solutions for:

1. **Linear**
2. **Square**

Each equation type is a class. Sometimes equations do not have real roots - in this case **throw your exception** and **handle** using try-catch-finally it.

Task 2

Take your first hometask, problem “Add 2 numbers”. Brush up the code, handle exceptions correctly and provide proper reactions (print) on situations:

1. *file not found*
2. *other file issues*
3. *parsing numbers*
4. *arithmetic overflow*

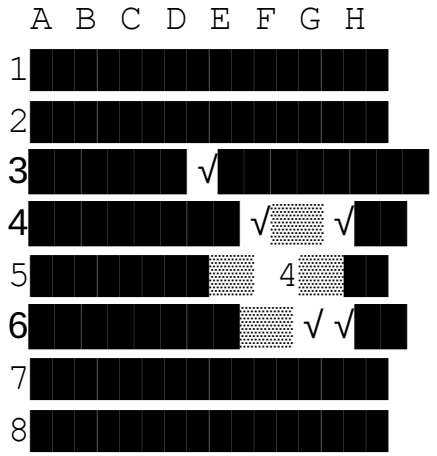
Task 3

For yesterday's implementation of the **Circle Equation** add exceptions.
Find best existing exception implementations for these cases.

Extra Task

Implement simple Miner game with console interface.

- Randomly set up bombs
- Handle bomb blast as exception.
- Move is made by typing coordinates (D4)
- Handle incorrect inputs
- Handle input for already processed cells



Home Task

Write the program, that calculated intersection point (class Point) of two sections (class Section). Handle following cases using exceptions mechanism: sections do not intersect (output - NO INTERSECTION), section(s) is degenerate (it's length is 0, output - DEGENERATE), sections coincide (COINCIDE). Add mandatory Input validation (INPUT ERROR).

E.g.
0 0 1 1 1 0 0 1

Answer
0.5 0.5