1495 UNIVERSITY OF ABERDEEN

JC2002 Java Programming

Lecture 13: Introduction to testing and dependency management

Learning objectives

- After today's session, you should be able to:
  - Explain the purpose of testing in software development
  - Follow test-driven software development process in your project
  - Manage dependencies in your projects

Good coding practice

- Following good coding practice helps to avoid mistakes and makes it easier to find possible bugs
  - Indent your code properly, use parenthesis and curly brackets consistently
  - Name your classes and variables logically and consistently
  - Make sure variables are always properly initialised
  - Use comments and optional annotations (such as `@Override`) to clarify any potential confusion in your program code
  - ***Test your code rigorously!***

Software testing

- Software testing is the process of verifying that the code is functioning correctly and producing the expected output
- Testing is an essential part of the software development life cycle
    - Testing ensures your program works as it is supposed to work
    - Testing helps to catch bugs and issues early in the development process, reducing the cost and time of fixing them later
    - Testing helps to keep software design modular so that the units can be tested in a meaningful way; if you are not able to test your class properly, your class design is maybe not appropriate

Levels of software testing

- Software can be tested at different levels, requiring different testing methodologies
  - **Unit testing**: testing units (classes or methods etc.) in isolation
  - **Integration testing**: testing two or more units together
  - **Functional testing**: testing specific functionality of an app
  - **System testing**: testing interfaces and functionality from end-user perspective
  - **System integration testing**: testing collaborating applications
- In this course, we focus on *unit testing*

Other types of software testing

- In this course, we focus on testing the code to make sure that it gives expected outputs and the program does not crash unexpectedly etc.; however, other types of tests also exist
  - **Performance testing**: testing on processor and network load, handling throughput, memory etc.
  - **Security testing**: testing application's robustness against security vulnerabilities and secure handling of data
  - **Usability testing**: testing that the applications is easy and logical to use for the end users
  - etc.

Writing unit tests

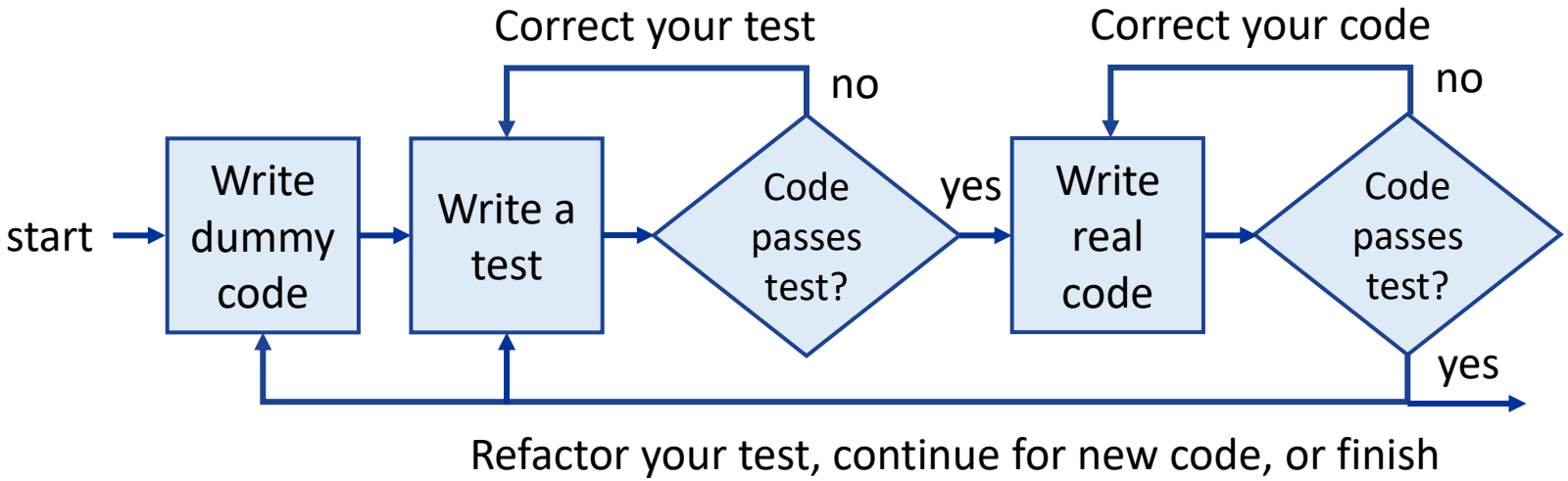
- Developers create test cases that cover all possible inputs and outputs of a particular unit of code
    - Test cases should include typical input, edge cases, unexpected behaviour, and handling of errors and exceptions
    - It can be time consuming and it may require significant amounts of effort from the developers!
- Unit tests are often automated using testing frameworks like Maven
    - Even though automated tests take more time for the initial setup, it saves time in a long run
    - In some cases, it is still the best option to run unit tests manually

Benefits of automated tests

- Automated testing helps reduce the time and effort required to test software applications, since the same tests can run multiple times without the need for human intervention
    - Automated tests run faster
    - Automated tests can be run more frequently
    - Automated tests free up testers to focus on more complex or exploratory testing tasks
    - Automated tests do not forget about what is left to test
    - Automated tests do not get bored!

Test-driven development

- Using tests to drive the implementation of the code



Example of test-driven development (1)

- Let us assume that we want to write a method to compute number of combinations, i.e.,

$$C(n,k) = n! / r!(n-r)!$$

Combinations.java

```
1  // Class for computing combinations
2  public class Combinations {
3    public int computeC(int n, int r) {
4      return -1; // indicates error
5    }
6  }
```

TestCombinations.java

```
1  public class TestCombinations {
2
3    // define test cases
4    static int[] n = {5,2,5,100};
5    static int[] r = {3,1,-1,50};
6
7    public static void main(String[] args) {
8
9      // run test cases
10     for(int i=0; i<n.length; i++) {
11       Combinations comb = new Combinations();
12       System.out.printf("n=%d, r=%d, C(n,r)=%d%n",
13         n[i],r[i],comb.computeC(n[i],r[i]));
14     }
15   }
16 }
```

Example of test-driven development (2)

Combinations.java

```
1  // Class for computing combinations
2  public class Combinations {
3    public int computeC(int n, int r) {
4      return -1; // indicates error
5    }
6  }
```

We first write a dummy method that always returns -1 (error)

Test fails, because it does not give the right answers

TestCombinations.java

```
1  public class TestCombinations {
2
3    // define test cases
4    static int[] n = {5,2,5,100};
5    static int[] r = {3,1,-1,50};
6
7    public static void main(String[] args) {
8
9      // run test cases
10     for(int i=0; i<n.length; i++) {
11       Combinations comb = new Combinations();
12       System.out.printf("n=%d, r=%d, C(n,r)=%d%n",
13         n[i],r[i],comb.computeC(n[i],r[i]));
14     }
15   }
16 }
```

```
n=5, r=3, C(n,r)=-1
n=2, r=2, C(n,r)=-1
n=5, r=-1, C(n,r)=-1
n=100, r=50, C(n,r)=-1
```

Example of test-driven development (3)

Combinations.java

```
1   // Class for computing combinations
2   public class Combinations {
3     private int fact(int x) {
4       int y = 1;
5       for(int i=1; i<=x; y *= i++);
6       return y;
7     }
8     public int computeC(int n, int r) {
9       int res = fact(n)/(fact(r)*fact(n-r));
10      return res; // return result
11    }
12  }
```

Write real functionality

TestCombinations.java

```
1   public class TestCombinations {
2
3     // define test cases
4     static int[] n = {5,2,5,100};
5     static int[] r = {3,1,-1,50};
6
7     public static void main(String[] args) {
8
9       // run test cases
10      for(int i=0; i<n.length; i++) {
11        Combinations comb = new Combinations();
12        System.out.printf("n=%d, r=%d, C(n,r)=%d%n",
13          n[i],r[i],comb.computeC(n[i],r[i]));
14      }
15    }
16  }
```

Example of test-driven development (4)

Combinations.java

```
1   // Class for computing combinations
2   public class Combinations {
3     private int fact(int x) {
4       int y = 1;
5       for(int i=1; i<=x; y *= i++);
6       return y;
7     }
8     public int computeC(int n, int r) {
9       int res = fact(n)/(fact(r)*fact(n-r));
10      return res; // return result
11    }
12  }
```

TestCombinations.java

```
1   public class TestCombinations {
2
3     // define test cases
4     static int[] n = {5,2,5,100};
5     static int[] r = {3,2,-1,50};
6
7     public static void main(String[] args) {
8
9       // run test cases
10      for(int i=0; i<n.length; i++) {
11        Combinations comb = new Combinations();
12        System.out.printf("n=%d, r=%d, C(n,r)=%d%n",
13          n[i],r[i],comb.computeC(n[i],r[i]));
14      }
15    }
16  }
```

```
n=5, r=3, C(n,r)=10
n=2, r=2, C(n,r)=1
n=5, r=-1, C(n,r)=0
Exception: / by zero
```

Correct results

Illegal input, should return -1!

Division by zero. Why is that?

Example of test-driven development (5)

Combinations.java

```
1  public class Combinations {
2    private long fact(long x) {
3      long y = 1;
4      for(long i=1; i<=x; y *= i++);
5      return y;
6    }
7    public long computeC(long n, long r) {
8      if(n<1 || r<1 || r>n || n>20) {
9        return -1; // illegal input
10     }
11     long res = fact(n)/(fact(r)*fact(n-r));
12     return res; // return result
13   }
14 }
```

Test for out-of-bounds input

Use **long** to handle larger values

TestCombinations.java

```
1  public class TestCombinations {
2
3    // define test cases
4    static int[] n = {5,2,5,100};
5    static int[] r = {3,1,-1,50};
6
7    public static void main(String[] args) {
8
9      // run test cases
10     for(int i=0; i<n.length; i++) {
11       Combinations comb = new Combinations();
12       System.out.printf("n=%d, r=%d, C(n,r)=%d%n",
13         n[i],r[i],comb.computeC(n[i],r[i]));
14     }
15   }
16 }
```

Example of test-driven development (6)

Combinations.java

```
1   public class Combinations {
2     private long fact(long x) {
3       long y = 1;
4       for(long i=1; i<=x; y *= i++);
5       return y;
6     }
7     public long computeC(long n, long r) {
8       if(n<1 || r<1 || r>n || n>20) {
9         return -1; // illegal input
10      }
11      long res = fact(n)/(fact(r)*fact(n-r));
12      return res; // return result
13    }
```

Now it works but note that max. value of *n* is 20 (even `long` cannot compute larger factorials); better implementations could be made!

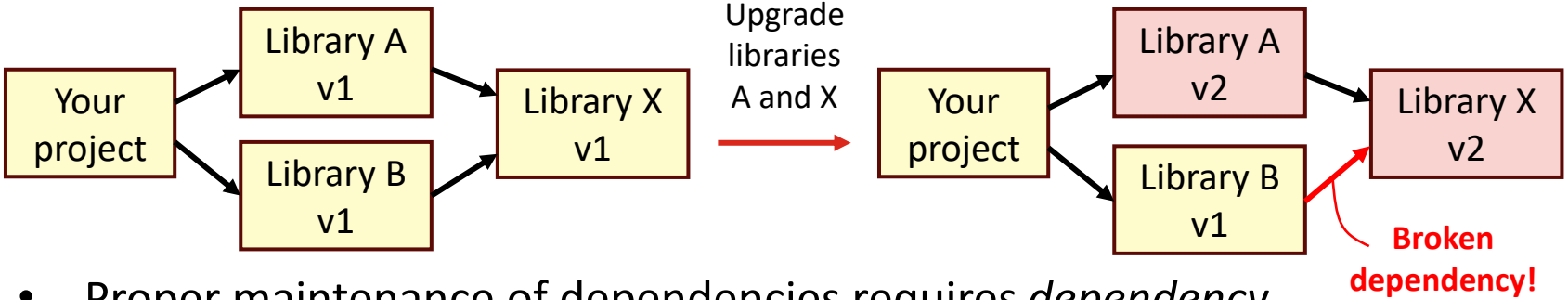
TestCombinations.java

```
1   public class TestCombinations {
2
3     // define test cases
4     static int[] n = {5,2,5,100};
5     static int[] r = {3,1,-1,50};
6
7     public static void main(String[] args) {
8
9       // run test cases
10      for(int i=0; i<n.length; i++) {
11        Combinations comb = new Combinations();
12        System.out.printf("n=%d, r=%d, C(n,r)=%d%n",
13          n[i],r[i],comb.computeC(n[i],r[i]));
14      }
15    }
16  }
```

```
n=5, r=3, C(n,r)=10
n=2, r=2, C(n,r)=1
n=5, r=-1, C(n,r)=-1
n=100, r=50, C(n,r)=-1
```

Dependency management

- Often large software project use different third-party libraries
  - There are different (sometimes complex) *dependencies* between libraries: certain library may use a specific version of another library
  - When upgrading to new versions, dependencies can get broken



- Proper maintenance of dependencies requires *dependency management*

Basics of dependency management

- When you upgrade a third-party library, you possibly need to make some changes in your code using that library
  - See the API documentation of the library
- In case of dependency-related errors, you need to find the library that produces the error
  - The error message often shows where the problem occurs, e.g.:

    ```
    java.lang.NoClassDefFoundError: ch/qos/logback/core/status/WarnStatus
    ```

  - Online tool serfish (https://serfish.com/jar/) can be useful for finding `.jar` packages for missing classes
  - We can also use dependency management tools

Questions, comments?