/*

pyetje nga libri Introduction to Software Testing;

Klasat e ushtrimeve mund te gjenden ne kete link: Programs for Introduction to Software Testing, edition 2, Ammann and Offutt ;

*/

Chapter 1

1. What is the difference between software fault and software failure?
   Solution. Software fault is often called a bug that servers as the cause of a failure. A failure is any deviation of the observed behavior from the expected behavior of the software.
2. The following exercise is intended to encourage you to think of testing in a more rigorous way than you may be used to. The exercise also hints at the strong relationship between specification clarity, faults, and test cases[1].
   (a) Write a Java method with the signature
   `public static Vector union (Vector a, Vector b)`
   The method should return a Vector of objects that are in either of the two argument Vectors.
   (b) Upon reflection, you may discover a variety of defects and ambiguities in the given assignment. In other words, ample opportunities for faults exist. Describe as many possible faults as you can. (*Note: Vector is a Java Collection class. If you are using another language, interpret Vector as a list.*)
   (c) Create a set of test cases that you think would have a reasonable chance of revealing the faults you identified above. Document a rationale for each test in your test set. If possible, characterize all of your rationales in some concise summary. Run your tests against your implementation.
   (d) Rewrite the method signature to be precise enough to clarify the defects and ambiguities identified earlier. You might wish to illustrate your specification with examples drawn from your test cases.
3. Below are four faulty programs. Each includes test inputs that result in failure. Answer the following questions about each program.

```java
/**
 * Find last index of element
 *
 * @param x array to search
 * @param y value to look for
 * @return last index of y in x; -1 if absent
 * @throws NullPointerException if x is null
 */
public int findLast (int[] x, int y)
{
  for (int i=x.length-1; i > 0; i--)
  {
    if (x[i] == y)
    {
      return i;
    }
  }
  return -1;
}
// test:  x = [2, 3, 5]; y = 2; Expected = 0
// Book website: FindLast.java
// Book website: FindLastTest.java
```

```java
/**
 * Find last index of zero
 *
 * @param x array to search
 *
 * @return last index of 0 in x; -1 if absent
 * @throws NullPointerException if x is null
 */
public static int lastZero (int[] x)
{
  for (int i = 0; i < x.length; i++)
  {
    if (x[i] == 0)
    {
      return i;
    }
  }
  return -1;
}
// test:  x = [0, 1, 0]; Expected = 2
// Book website: LastZero.java
// Book website: LastZeroTest.java
```

```java
/**
 * Count positive elements
 *
 * @param x array to search
 * @return count of positive elements in x
 * @throws NullPointerException if x is null
 */
public int countPositive (int[] x)
{
  int count = 0;
  for (int i=0; i < x.length; i++)
  {
    if (x[i] >= 0)
    {
      count++;
    }
  }
  return count;
}
// test:  x = [-4, 2, 0, 2]; Expected = 2
// Book website: CountPositive.java
// Book website: CountPositiveTest.java
```

```java
/**
 * Count odd or positive elements
 *
 * @param x array to search
 * @return count of odd/positive values in x
 * @throws NullPointerException if x is null
 */
public static int oddOrPos(int[] x)
{
  int count = 0;
  for (int i = 0; i < x.length; i++)
  {
    if (x[i]%2 == 1 || x[i] > 0)
    {
      count++;
    }
  }
  return count;
}
// test:  x = [-3, -2, 0, 1, 4]; Expected = 3
// Book website: OddOrPos.java
// Book website: OddOrPosTest.java
```

(a) Explain what is wrong with the given code. Describe the fault precisely by proposing a modification to the code.
(b) If possible, give a test case that does **not** execute the fault. If not, briefly explain why not.
(c) If possible, give a test case that executes the fault, but does **not** result in an error state. If not, briefly explain why not.
(d) If possible give a test case that results in an error, but **not** a failure. If not, briefly explain why not. Hint: Don't forget about the program counter.
(e) For the given test case, describe the first error state. Be sure to describe the complete state.
(f) Implement your repair and verify that the given test now produces the expected output. Submit a screen printout or other evidence that your new program works.

4. Consider the following three example classes. These are OO faults taken from Joshua Bloch's Effective Java, Second edition. Answer the following questions about each.

```
class Vehicle implements Cloneable
{
   private int x;
   public Vehicle (int y) { x = y;}
   public Object clone()
   {
      Object result = new Vehicle (this.x);
      // Location "A"
      return result;
   }
   // other methods omitted
}
class Truck extends Vehicle
{
   private int y;
   public Truck (int z) { super (z); y = z;}
   public Object clone()


    {
      Object result = super.clone();
      // Location "B"
      ((Truck) result).y = this.y;  // throws ClassCastException
      return result;
    }
    // other methods omitted
 }
 //Test: Truck suv = new Truck (4); Truck co = suv.clone()
 //      Expected: suv.x = co.x; suv.getClass() = co.getClass()
```

Note: Relevant to Bloch, Item 11 page 54.
Book website: Vehicle.java, Truck.java, CloneTest.java

```
public class BigDecimalTest
{
 BigDecimal x = new BigDecimal ("1.0");
 BigDecimal y = new BigDecimal ("1.00");
 // Fact:  !x.equals (y), but x.compareTo (y) == 0

 Set <BigDecimal> BigDecimalTree = new TreeSet <BigDecimal> ();
 BigDecimalTree.add (x);
 BigDecimalTree.add (y);
 // TreeSet uses compareTo(), so BigDecimalTree now has 1 element

 Set <BigDecimal> BigDecimalHash = new HashSet <BigDecimal> ();
 BigDecimalHash.add (x);
 BigDecimalHash.add (y);
 // HashSet uses equals(), so BigDecimalHash now has 2 elements
}
 // Test: System.out.println ("BigDecimalTree = " + BigDecimalTree);
       // System.out.println ("BigDecimalHash = " + BigDecimalHash);
       // Expected: BigDecimalTree = 1; BigDecimalHash = 1

// See Java Doc for add() in Set Interface
// The problem is that in BigDecimal, equals() and compareTo()
// are inconsistent. Let's suppose we decide that compareTo() is correct,
// and that equals()is faulty.
```

Note: Relevant to Bloch, Item 12 page 62.
Book website: class BigDecimalTest.java

```
class Point
{
   private int x; private int y;
   public Point (int x, int y) { this.x=x; this.y=y; }

   @Override public boolean equals (Object o)
   {
      // Location A
      if (!(o instanceof Point)) return false;
      Point p = (Point) o;
      return (p.x == this.x) && (p.y == this.y);
   }
}
class ColorPoint extends Point
{
   private Color color;
   // Fault: Superclass instantiable; subclass state extended

   public ColorPoint (int x, int y, Color color)
   {
      super (x,y);
      this.color = color;
   }
```

```
@Override public boolean equals (Object o)
{
  // Location B
  if (!(o instanceof ColorPoint)) return false;
  ColorPoint cp = (ColorPoint) o;
  return (super.equals(cp) && (cp.color == this.color));
}
// Tests:
  Point p  = new Point (1,2);
  ColorPoint cp1   = new ColorPoint (1,2,RED);
  ColorPoint cp2   = new ColorPoint (1,2,BLUE);
  p.equals (cp1);    // Test 1: Result = true;
  cp1.equals (p);    // Test 2: Result = false;
  cp1.equals (cp2);  // Test 3: Result = false;
// Expected: p.equals (cp1) = true; cp1.equals (p) = true,
//       cp1.equals (cp2) = false
```

Note: Relevant to Bloch, Item 17 page 87.
Book website: Point.java, ColorPoint.java, PointTest.java

(a) Explain what is wrong with the given code. Describe the fault precisely by proposing a modification to the code.
(b) If possible, give a test case that does **not** execute the fault. If not, briefly explain why not.
(c) If possible, give a test case that executes the fault, but does **not** result in an error state. If not, briefly explain why not.
(d) If possible give a test case that results in an error, but **not** a failure. If not, briefly explain why not. Hint: Don't forget about the program counter.
(e) In the given code, describe the first error state. Be sure to describe the complete state.
(f) Implement your repair and verify that the given test now produces the expected output. Submit a screen printout or other evidence that your new program works.

Chapter 2

1. How are fault and failures related to testing and debugging?

Chapter 3

1. Develop JUnit tests for the BoundedQueue class. (at the books website) Make sure your tests check every method.
2. The following JUnit test method for the sort( ) method has a non-syntatic flaw. Find the flaw and describe it in terms of the RIPR model. Be as precise, specific, and concise as you can. In the test method, names is an instance of an object that stores strings and has methods add( ), sort( ), and getFirst( ), which do exactly what you would expect from their names.

```
@Test
public void testSort()
{
  names.add ("Laura");
```

```
  names.add ("Han");
  names.add ("Alex");
  names.add ("Ashley");
  names.sort();
  assertTrue ("Sort method", names.getFirst().equals ("Alex"));
}
```

3.

Consider the following example class. PrimeNumbers has three methods. The first, computePrimes(), takes one integer input and computes that many prime numbers. iterator() returns an Iterator that will iterate through the primes, and toString() returns a string representation.

```
public class PrimeNumbers implements Iterable<Integer>
{
  private List<Integer> primes = new ArrayList<Integer>();

  public void computePrimes (int n)
  {
    int count = 1; // count of primes
    int number = 2; // number tested for primeness
    boolean isPrime; // is this number a prime
    while (count <= n)
    {
      isPrime = true;
      for (int divisor = 2; divisor <= number / 2; divisor++)
      {
        if (number % divisor == 0)
        {
          isPrime = false;
          break; // for loop
        }
      }
      if (isPrime && (number % 10 != 9)) // FAULT
      {
        primes.add (number);
        count++;
      }
      number++;
    }
  }
```

```
  @Override public Iterator<Integer> iterator()
  {
    return primes.iterator();
  }
```



```
  @Override public String toString()
  {
    return primes.toString();
  }
}
```

computePrimes() has a fault that causes it **not** to include prime numbers whose last digit is 9 (for example, it omits 19, 29, 59, 79, 89, 109, ...). If possible, describe five tests. You can describe the tests as sequences of calls to the above methods, or briefly describe them in words. Note that the last two tests require the test oracle to be described.

(a) A test that does not reach the fault
(b) A test that reaches the fault, but does not infect
(c) A test that infects the state, but does not propagate
(d) A test that propagates, but does not reveal
(e) A test that reveals the fault

If a test cannot be created, explain why.

Chapter 4

Find a refactoring in some large, existing system. Build tests that capture the behavior relevant to that part of the system. Refactor, and then check that the tests still pass.

1 **Describe, with examples, the way in which a defect in software can cause harm to a person, to the environment or to a company. (K2)**
2 **Distinguish between the root cause of a defect and its effects. (K2)**
3 **Give reasons why testing is necessary by giving examples. (K2)**
4 **Describe why testing is part of quality assurance and give examples of how testing contributes to higher quality. (K2)**
5 **Recall the terms 'mistake', 'defect', 'fault', 'failure' and the correspon ding terms 'error' and 'bug'. (Kl)**
6 **Explain the fundamental principles in testing. (K2)**

1 **Recall the common objectives of testing. (Kl)**
2 **Describe the purpose of testing in software development, maintenance and operations as a means to find defects, provide confidence and infor mation, and prevent defects. (K2)**

EXERCISE: TEST PSYCHOLOGY

Read the email below, and see what clues you find to help you identify problems in the scenario described. Categorize the clues/problems as:

- possible people, psychology and attitude problems;
- other problems, e.g. possible test management and role problems, possible product problems.

Hi there!
Well, I nearly caused a panic today because I thought I had found a mega showstopper on the trading system we are testing. The test manager and others got involved examining databases first on the server and then on the gateway that feeds the clients, checking update logs from processes that ran overnight as well as checking data passed to the client. Eventually I found the problem. I had mis-clicked on a .bat file when running up a client and had run up the wrong client environment. By that time the test manager was ready to say a few short words in my ear, particularly as the development people had started to get involved and they have zero tolerance for mistakes made by testers. The only saving grace was that I found the mistake and not one of the developers.

It was, objectively, an interesting mistake. When you log into the server test environments, the panels always show the environment to which you are connected. In our case we have two test environments called Systest14 and Systest15 and my tests were set up in Systest15. To run up the clients, we have to run .bat files for either a 14 or 15 client. I had started two clients, that is two exchange participants, so I could do some trading between them.

It appears I started the first client OK in environment 15 but when I started the second, I accidentally moved the mouse a fraction so it ran the 14 .bat file that is next to it in the Explorer file list. To make matters worse, the client screens do not show the environment to which you are attached.

At first I felt a bit stupid having caused much hectic and wasted activity. On reflection I thought that if I, as a reasonably competent person, can make a mistake like this then something is wrong. On the server side when I log on to a test environment, I have to enter the environment name and it's shown on all the panels. On the client side, I run a client test environment by selecting a .bat file from a list of many and have to ensure I click on the right file. There is neither a display nor the ability to determine the client environment in which I am working.

So I am going to log this as a high priority, or even showstopper, error - the client does not show the environment. In real life terms, it means a real user could be connected to the production system and think he is connected to a test system and screw up trading. I know this happened once on the equities trading system, when a trader entered a load of test transactions into the production system by mistake and caused mayhem.

As an addendum to this story, a couple of days later one of the testers found what appeared to be another mega showstopper. He and the test manager spent three hours crawling all over the system before they discovered the 'error'. A new filter had been added to the client software to filter transactions displayed in panels by geographical market. Unknown to them, it was set to a default of the German market, whereas they thought they were in the UK market. Consequently, at first sight, it appeared there were fundamental problems with the network transaction bus and the message-broadcasting systems. Apart from the issue that they should have been informed of this change, it raised a similar problem to the one I had experienced -the client system does not display the market in which you are trading.

Well - I'm off for another happy day at the
office! All the best

EXERCISE SOLUTION

People, psychology and attitude problems include, for example:

- Poor relationships between the test team and the test manager, and the testers and developers, e.g. 'By that time the test manager was ready to say a few short words in my ear, particularly as the development people had started to get involved and they have zero tolerance for mistakes made by testers. The only saving grace was that I found the mistake and not one of the developers.'
- Emotive use of language - understandable in the circumstances but not suitable for reporting problems, e.g. 'Well, I nearly caused a panic today because I thought I had found a mega showstopper on the trading system we are testing,' and 'As an addendum to this story, a couple of days later one of the testers found what appeared to be another mega-showstopper.'
- Initial diffidence overcome by revisiting the problem - if one person can make this mistake then others will. 'At first I felt a bit stupid having caused much hectic and wasted activity. On reflection I thought that if I, as a reasonably competent person, can make a mistake like this then something is wrong.'
- Understandable use of sarcasm ... 'Well - I'm off for another happy day at the office!'

Other problems include test management and role problems, for example:

- Configuration management and release control - A new filter had been added to the client software to filter transactions displayed in panels by geographical market.'
- Configuration management, relationships, communications - Apart from the issue that they should have been informed of this change ....'
- Does the test manager really understand his role? 'He and the test manager spent three hours crawling all over the system before they discovered the "error",' and 'The test manager and others got involved examining databases.'

There are some product problems, although no functionality or technical problems. Not all the problems we encounter as testers are functionality or technical problems. There are some non-functional problems - specifically, usability - which indicate that a real user might be inconvenienced or worse by this problem:

- 'I had mis-clicked on a .bat file ...'
- 'In real life terms, it means a real user could be connected to the production system and think he is connected to a test system and screw up trading. I know this happened once ... when a trader entered a load of test transactions into the production system by mistake and caused mayhem.'
- 'It raised a similar problem to the one I had experienced - the client system does not display the market in which you are trading.'
- 'There is neither a display nor the ability to determine the client environment in which I am working.' And 'To make matters worse, the client screens do not show the environment to which you are attached.'
- 'Unknown to them, it was set to a default of the German market, whereas they thought they were in the

3. Answer the following questions for the method search() below:

```
public static int search (List list, Object element)
// Effects: if list or element is null throw NullPointerException
//   else if element is in the list, return an index
//   of element in the list; else return -1
//   for example, search ([3,3,1], 3) = either 0 or 1
//       search ([1,7,5], 2) = -1
```

Base your answer on the following characteristic partitioning:

Characteristic: Location of element in list
- Block 1: element is first entry in list
- Block 2: element is last entry in list
- Block 3: element is in some position other than first or last

(a) "Location of element in list" fails the disjointness property. Give an example that illustrates this.
(b) "Location of element in list" fails the completeness property. Give an example that illustrates this.
(c) Supply one or more new partitions that capture the intent of "Location of element in list" but do not suffer from completeness or disjointness problems.

Pyetje/Alternativa:

1. What is Software Testing?
   a) Analyzing documentation
   b) Writing code
   c) Executing a system to identify bugs and errors
   d) Compiling source code

2. What is a test case?
   a) A list of all the bugs in the software
   b) A document that describes how a test should be executed
   c) A report of testing results
   d) The final version of the software

3. What is Unit testing?
   a) Testing individual components in isolation
   b) Testing the system's performance
   c) Testing database interactions
   d) Testing the user interface

4. What is Integration testing?
   a) Testing database queries
   b) Testing interactions between multiple modules
   c) Testing a system's overall functionality
   d) Testing UI responsiveness

5. What is System testing?
   a) Performing end-to-end checks of the entire system
   b) Testing integration interactions
   c) Testing database schema
   d) Testing individual lines of code

6. Which of the following is NOT a principle of software testing?
   a) Testing shows the presence of defects
   b) Testing is best done only at the end of development
   c) Early testing saves time and cost
   d) Exhaustive testing is impossible

   Testing should be performed early in the development process to catch defects early, rather than waiting until the end.

7. Which of the following is true about software testing?
   a) Testing only focuses on fuctional requirements
   b) Software testing guarantees that the software is bug-free
   c) Software testing is an ongoing activity throughout the development lifecycle
   d) Software testing is not needed after deployment

8. What is the difference between verification and validation?
   a) Verification checks if the product is built correctly, validation checks if the product is correct
   b) Verification is testing, validation is documentation
   c) Verification is about testing the product in the real world, validation is done by developers
   d) There is no difference

9. What is a bug in software testing?
   a) A testing method
   b) A feature of the software

c) A malfunction or flaw in the software
d) A software update

10. What is Regression testing?
    a) Testing only new features
    b) Conducting user interface checks
    c) Validating database constraints
    d) Re-testing existing code to ensure new changes don't affect functionality

11. What is Functional testing?
    a) Checking database constraints
    b) Validating that features work as expected according to requirements
    c) Analyzing code logic
    d) Testing the system's performance

12. What is Stress testing?
    a) Evaluating system performance under extreme loads
    b) Checking user interface responsiveness
    c) Testing for system scalability
    d) Analyzing database query optimization

    Stress Testing evaluates system performance under high loads, extreme traffic, and stress conditions. It helps identify performance bottlenecks and ensures the system remains stable even under unusual pressures.

13. What is ad-hoc testing?
    a) Automated test planning
    b) Informal testing without pre-planning
    c) Testing only in a simulated environment
    d) Structured test cases

    Ad-hoc testing is an informal method where testers explore the system to find bugs without any formal test planning.

14. What is white-box testing?
    a) Testing done by external users
    b) Testing the user interface design
    c) Testing the security features of the software
    d) Testing based on the internal structure of the software

15. What is Black-box testing?
    a) Testing user interface components
    b) Compiling application source files
    c) **Testing with no access to source code**
    d) Testing only backend interactions

16. What is test coverage?
    a) Percentage of code or features tested
    b) Number of test cases executed
    c) The speed of test execution
    d) Number of bugs identified

17. What is a defect life cycle?
    a) The process of resolving defects
    b) The various stages a defect goes through from identification to resolution
    c) The stages of deployment in the software lifecycle
    d) The process of adding new features to the software

18. What is automated testing?
    a) Testing performed by the end-users
    b) Testing executed using software tools and scripts
    c) Testing done manually by testers
    d) Testing the system under extreme conditions

19. What is manual testing?
    a) Testing performed without tools or automation scripts
    b) Testing using programming languages
    c) Testing focused on automation
    d) Testing done only on mobile devices

20. What does the V-model in software testing represent?
    a) Testing is skipped
    b) Testing and development are executed simultaneously
    c) Testing begins as soon as the requirements are defined
    d) Testing is done after development

The V-model is a software development methodology where testing starts as soon as the requirements are defined, ensuring parallel development and testing.

21. What is the first level of testing usually performed on software?
    a) Integration Testing
    b) System Testing
    c) Acceptance Testing
    d) Unit Testing

22. Which of the following tools is commonly used for unit testing?
    a) LoadRunner
    b) Selenium
    c) JUnit
    d) JIRA

23. What does boundary value analysis focus on?
    a) Ensuring complete code coverage
    b) Testing at the boundary of input ranges
    c) Testing all input combinations
    d) Testing invalid inputs only

24. What is the main principle of Agile Testing?
    a) Automating all tests
    b) Focusing only on functional testing
    c) Testing during every phase of the development cycle
    d) Testing at the end of development

25. What are Test Oracles?
    a) A method to validate expected outcomes against actual results
    b) Compilation error logs
    c) Test automation tools
    d) Database query log

26. Which of these is NOT a level of software testing?
    a) Integration Testing
    b) System Testing
    c) Unit Testing

d) Error Testing

27. What does a test case in Black Box Testing include?
    a) Input data and expected output
    b) Compiler logs
    c) Source code lines
    d) Memory allocation details

28. What is the main benefit of Black Box testing?
    a) Optimizing database queries
    b) Testing from the end-user perspective
    c) Compiling source code faster
    d) Finding bugs in code syntax

29. What type of testing is typically used for Black Box testing?
    a) Integration Testing
    b) Unit Testing
    c) Regression Testing
    d) Functional Testing

30. Which of the following is a disadvantage of Black Box testing?
    a) Inability to handle large datasets
    b) Limited test coverage
    c) No feedback on internal system logic
    d) Slow execution speed

31. What is an example of a test case that could be created using Black Box testing?
    a) Analyzing the system's memory consumption
    b) Testing a login form to verify valid/invalid input handling
    c) Testing code for handling exceptions
    d) Debugging the system's algorithms

32. What is the main difference between Black Box testing and White Box testing?
    a) Black Box Testing focuses on internal code; White Box Testing focuses on external behavior
    b) There is no difference between the two testing types
    c) White Box Testing is for functional testing; Black Box Testing is for security testing

d) Black Box Testing does not require knowledge of code; White Box Testing does

33. Which of the following is a common technique used in white box testing?
    a) Boundary value analysis
    b) Random testing
    c) Path testing
    d) Usability testing
    Path testing is a common white box testing technique where all possible paths through the code are tested to ensure complete coverage

34. What is Code Coverage in White Box testing?
    a) The time it takes for the code to execute
    b) The process of ensuring all external interfaces are tested
    c) The process of removing unnecessary code
    d) The percentage of code executed during testing

35. What is the role of a test case in white box testing?
    a) To verify the logic and structure of the code
    b) To evaluate system performance under load
    c) To identify the user interface issues
    d) To check the system's interaction with hardware

36. What type of errors can White Box testing help identify?
    a) User Interface bugs
    b) Logical errors and security flaws
    c) Syntax errors
    d) Only memory leaks

37. What challenges do testers face during White Box testing?
    a) Limited access to external interactions
    b) Complexity of code
    c) Requires advanced technical skills
    d) All of the mentioned

/* pyetje nga libri Effective Software Testing */

Exercise 3.2. See the `remove` method below:

```
public boolean remove(Object o) {
  if (o == null) {
    for (Node<E> x = first; x != null; x = x.next) {
      if (x.item == null) {
        unlink(x);
        return true;
      }
    }
  } else {
    for (Node<E> x = first; x != null; x = x.next) {
      if (o.equals(x.item)) {
        unlink(x);
        return true;
      }
    }
  }
  return false;
}
```

This is the implementation of JDK8's LinkedList remove method.

Create a test suite (i.e. a set of tests) that achieves 100% line coverage. Use as few tests as possible. Feel free to write them as JUnit tests or simply as a set of inputs and expected outputs.

Exercise 3.4. Consider the expression (A & B) | C with the truth table:

Test case	A	B	C	(A & B) | C
1	T	T	T	T
2	T	T	F	T
3	T	F	T	T
4	T	F	F	F
5	F	T	T	T
6	F	T	F	F
7	F	F	T	T
8	F	F	F	F

What test suite(s) achieve 100% MC/DC? The numbers correspond to the test case column in the truth table. Select all that apply.

A. {2, 3, 4, 6}
B. {2, 4, 5, 6}
C. {1, 3, 4, 6}
D. {3, 4, 5, 8}

Exercise 3.6. See the method below:

```
public String sameEnds(String string) {
  int length = string.length();
  int half = length / 2;

  String left = "";
  String right = "";

  int size = 0;
  for (int i = 0; i < half; i++) {
    left = left + string.charAt(i);
    right = string.charAt(length - 1 - i) + right;

    if (left.equals(right)) {
      size = left.length();
    }
  }

  return string.substring(0, size);
}
```

Which of these properties is the major advantage of unit tests over integration tests?

- ○ Fast
- ○ Reliable
- ○ Isolate failures
- ○ Simulate users

Which of these properties is the major advantage of unit tests over integration tests?

- ○ Fast
- ○ Reliable
- ● Isolate failures ✔
- ○ Simulate users

Explanation

Integration tests and unit tests can both be fast. Neither really simulate users in the same way that Acceptance tests or even Systems tests might. Unit tests do a very good job of isolating failures: it's easy to look at a failing unit test and investigate the source of the error. Integration tests incorporate multiple modules or classes, and may not test the edge case of every method of all classes, so are not as finely-grained.

Exercise 6.3. You are testing a system that triggers advanced events based on complex combinations of external, boolean conditions relating to the weather (outside temperature, amount of rain, wind, etc). The system has been designed cleanly and consists of a set of co-operating classes that each have a single responsibility. You use specification-based testing for this logic and decide to test it using mocks.

Which of the following is a valid test strategy?

A. You use mocks to support observing the external conditions.
B. You create mock objects to represent each variant you need to test.
C. You use mocks to control the external conditions and to observe the event being triggered.
D. You use mocks to control the triggered events.

Solution. C

Exercise 6.4. Class A depends on a static method in another class B. If you want to test class A, which of the following two actions should you apply to do this properly?

- **Approach 1:** Mock class B to control the behavior of the methods in class B.
- **Approach 2:** Refactor class A, so the outcome of the method of class B is now used as a parameter.

A. Only approach 1.
B. Neither.
C. Only approach 2.
D. Both.

Solution. Both. Megjithate zgjidhje me e mire do ishte refactor duke shtuar dependency injection nese klasa B nuk eshte ndonje external infrastructure ose nuk komunikon me databazen.

Exercise 6.6. According to the guidelines provided in the book, what types of classes should we mock and what should we not mock?

Solution. What to mock:

- Dependencies that are too slow. If a dependency is too slow, for any reason, it might be a good idea to simulate that dependency. (p.sh classes that deal with databases or web services)

- Dependencies that communicate with external infrastructure. If the dependency talks to (external) infrastructure, it might be too slow or too complex to set up the required infrastructure.
- Hard to simulate cases: For example, when we would like the dependency to throw an exception. Forcing an exception might be tricky when using the real dependency but easy to do with a stub.

What NOT to mock:

- Entities.
- Native libraries and utitility methods. (sepse jane klasa te gatshme dhe i marrim si te sakte, kujto gabimin me TextArea qe kemi patur tek detyra e kursit)
- Things that are simple enough. If you feel the class is too simple to be mocked, then it probably is.

Exercise 7.3: How can we improve the testability of the OrderDeliveryBatch class below?

```
public class OrderDeliveryBatch {

  public void runBatch() {

    OrderDao dao = new OrderDao();
    DeliveryStartProcess delivery = new DeliveryStartProcess();

    List<Order> orders = dao.paidButNotDelivered();

    for (Order order : orders) {
      delivery.start(order);

      if (order.isInternational()) {
        order.setDeliveryDate("5 days from now");
      } else {
        order.setDeliveryDate("2 days from now");
      }
    }
  }
}

class OrderDao {
  // accesses a database
}

class DeliveryStartProcess {
  // communicates with a third-party webservice
}
```

Exercise 7.4: Consider the KingsDayDiscount below:

```
public class KingsDayDiscount {

  public double discount(double value) {

    Calendar today = Calendar.getInstance();

    boolean isKingsDay = today.get(MONTH) == Calendar.APRIL
        && today.get(DAY_OF_MONTH) == 27;

    return isKingsDay ? value * 0.15 : 0;

  }
}
```

What can we do to improve the testability of this class?

Question 3 Put the test cases that implement the following test conditions into the best order for the test execution schedule, for a test that is checking modifications of customers on a database.

1 Print modified customer record.
2 Change customer address: house number and street name.
3 Capture and print the on-screen error message.
4 Change customer address: postal code.
5 Confirm existing customer is on the database by opening that record.
6 Close the customer record and close the database.
7 Try to add a new customer with no details at all.

a. 5,4, 2,1, 3, 7, 6
b. 4,2,5,1,6,7,3
c. 5,4,2,1,7,3,6
d. 5,1, 2, 3,4, 7, 6

Question 13 If you are flying with an economy ticket, there is a possibility that you may get upgraded to business class, especially if you hold a gold card in the airline's frequent flier program. If you don't hold a gold card, there is a possibility that you will get 'bumped' off the flight if it is full and you check in late. This is shown in Figure 4.5. Note that each box (i.e. statement) has been numbered.

Three tests have been run:

Test 1: Gold card holder who gets upgraded to business class

Test 2: Non-gold card holder who stays in economy

Test 3: A person who is bumped from the flight

What is the statement coverage of these three tests?

a. 60%

b. 70%

c. 80%

d. 90%

Equivalence Partitioning/Boundary Value Analysis exercise

Scenario: If you take the train before 9:30 am or in the afternoon after 4:00 pm until 7:30 pm ('the rush hour'), you must pay full fare. A saver ticket is available for trains between 9:30 am and 4:00 pm, and after 7:30 pm.

What are the partitions and boundary values to test the train times for ticket types? Which are valid partitions and which are invalid partitions? What are the boundary values? (A table may be helpful to organize your partitions and boundaries.) Derive test cases for the partitions and boundaries.

Are there any questions you have about this 'requirement'? Is anything unclear?

EP/BVA exercise

The first thing to do is to establish exactly what the boundaries are between the full fare and saver fare. Let's put these in a table to organize our thoughts:

Scheduled departure time	≦ 9:29 am	9:30 am – 4:00 pm	4:01 pm – 7:30 pm	≧ 7:31 pm
Ticket type	full	saver	full	saver

We have assumed that the boundary values are: 9:29 am, 9:30 am, 4:00 pm, 4:01 pm, 7:30 pm and 7:31 pm. By setting out exactly what we think is meant by the specification, we may highlight some ambiguities or, at least, raise some questions - this is one of the benefits of using the technique! For example:

'When does the morning rush hour start? At midnight? At 11:30 pm the previous day? At the time of the first train of the day? If so, when is the first train? 5:00 am?'

This is a rather important omission from the specification. We could make an assumption about when it starts, but it would be better to find out what is correct.

- If a train is due to leave at exactly 4:00 pm, is a saver ticket still valid?
- What if a train is due to leave before 4:00 pm but is delayed until after 4:00 pm? Is a saver ticket still valid? (i.e. if the actual departure time is different to the scheduled departure time)

Our table above has helped us to see where the partitions are. All of the partitions in the table above are valid partitions. It may be that an invalid partition would be a time that no train was running, e.g. before 5:00 am, but our specification didn't mention that! However it would be good to show this possibility also. We could be a bit more formal by listing all valid and invalid partitions and boundaries in a table, as we described in Section 4.3.1, but in this case it doesn't actually add a lot, since all partitions are valid.

Here are the test cases we can derive for this example:

Test case reference	**Input**	**Expected outcome**
1	Depart 4:30 am	Pay full fare
2	Depart 9:29 am	Pay full fare
3	Depart 9:30 am	Buy saver ticket
4	Depart 11:37 am	Buy saver ticket
5	Depart 4:00 pm	Buy saver ticket
6	Depart 4:01 pm	Pay full fare
7	Depart 5:55 pm	Pay full fare
8	Depart 7:30 pm	Pay full fare
9	Depart 7:31 pm	Buy saver ticket
10	Depart 10:05 pm	Buy saver ticket

Note that test cases 1, 4, 7 and 10 are based on equivalence partition values; test cases 2, 3, 5, 6, 8 and 9 are based on boundary values. There may also be other information about the test cases, such as preconditions, that we have not shown here.

Statement and decision testing exercise

Scenario: A vending machine dispenses either hot or cold drinks. If you choose a hot drink (e.g. tea or coffee), it asks if you want milk (and adds milk if required), then it asks if you want sugar (and adds sugar if required), then your drink is dispensed.

a. Draw a control flow diagram for this example. (Hint: regard the selection of the type of drink as one statement.)

b. Given the following tests, what is the statement coverage achieved? What is the decision coverage achieved?

   Test 1: Cold drink

   Test 2: Hot drink with milk and sugar

c. What additional tests would be needed to achieve 100% statement coverage? What additional tests would be needed to achieve 100% decision coverage?

Statement and decision testing exercise

The control flow diagram is shown in Figure 4.8. Note that drawing a control diagram here illustrates that structural testing can also be applied to the structure of general processes, not just to computer algorithms. Flowcharts are generally easier to understand than text when you are trying to describe the results of decisions taken on later events.

On Figure 4.9, we can see the route that Tests 1 and 2 have taken through our control flow graph. Test 1 has gone straight down the left-hand side to select a cold drink. Test 2 has gone to the right at each opportunity, adding both milk and sugar to a hot drink.

Every statement (represented by a box on the diagram) has been covered by our two tests, so we have 100% statement coverage.

We have not taken the No exit from either the 'milk?' or 'sugar?' decisions, so there are two decision outcomes that we have not tested yet. We did test both of the outcomes from the 'hot or cold?' decision, so we have covered four out of six decision outcomes. Decision coverage is 4/6 or 67% with the two tests.

No additional tests are needed to achieve statement coverage, as we already have 100% coverage of the statements.

One additional test is needed to achieve 100% decision coverage:

Test 3: Hot drink, no milk, no sugar

This test will cover both of the 'No' decision outcomes from the milk and sugar decisions, so we will now have 100% decision coverage.

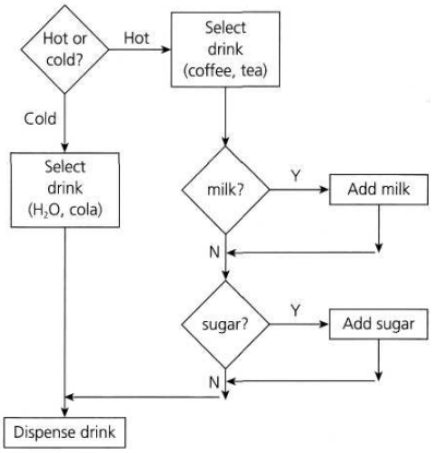

FIGURE 4.8 Control flow diagram for drinks dispenser

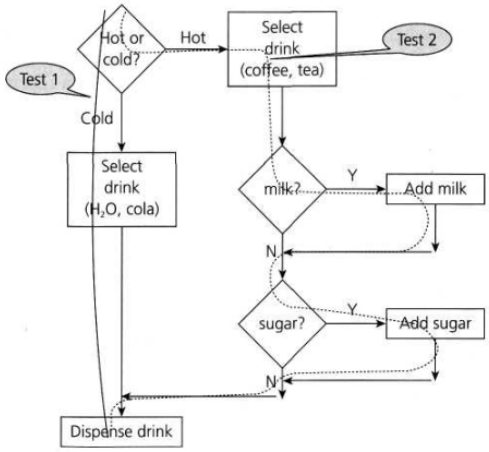

FIGURE 4.9 Control flow diagram showing coverage of tests

Exercise 8.4. It is time to practice TDD. A very common problem used to practice is to calculate the final score of a bowling game.

In bowling, a game consists of 10 rounds. In each round, each player has a "frame". In a frame the player can make two attempts to roll over 10 pins with the bowling ball. The score for each frame is the number of pins knocked down, with a bonus for a strike or a spare.

In a strike, the player knocks over all pins with one roll. On top of the 10 points, the player also gets a bonus: the total number of pins knocked over in the next frame. Take the following rolls as an example: `[X] [1 2]` (each set of `[ ]` is one frame, `X` indicates a strike). The player has accumulated a total of 16 points in these two frames. The first frame is 10+3 points (10 for the strike, 3 is the sum of the next two rolls 1+2), and the second frame is simply 3 (the sum of the rolls).

For a spare, the player has to knock down all pins within one frame (i.e., two rolls). As a bonus, the points for the next roll are added to the score of the frame. As an example take `[4 /] [3 2]` (`/` represents a spare). The player scores 13 points for the first frame (10 pins + 3 from the next roll), plus 5 for the second frame, for a total of 18 points.

If a strike or a spare is achieved in the tenth (and final) frame, the player has a right for an additional 1 (for a spare) or 2 (for a strike) rolls. However, the total rolls for this frame cannot exceed 3 (i.e., rolling a strike with one of the extra rolls does not grant more rolls).

Write a program that receives the results of the 10 frames and returns the final score of the game. Use the TDD cycle: write a test, make it pass, repeat.