Task 1(a)

Consider the following unit test in Java

```
public class MyTest {
    private MyClass obj = new MyClass();

    @Test
    public void test1()
    {
        assertTrue(obj.checkValue(10));
        assertFalse(obj.checkValue(15));
        assertTrue(obj.checkValue(20));
    }
}
```

Assuming that the test passes, describe as many details of class MyClass as possible.

MyClass contains a method called '**checkValue**', this method will take an int as an argument, and then return a boolean value that is based on the input argument value.

- When the input to **MyClass** is 10 or 20, the method will return true.
- When the input to **MyClass** is 15, the method will return false.

Task 1(b)

Describe the concept of unit testing. What can it be used for and how does it differ from other kinds of testing?

A unit test is an automated test with the purpose of ensuring that a specific section of code will function as intended. Unit tests are a good way of testing certain sections of a program, where the developer will provide the program with a set of data that it must use. This type of testing is a good method for identifying bugs in the code, as opposed to other types of testing such as functional testing, which will test the entire program.

Unit testing is one of the simplest forms of testing, and therefore can usually be carried out at any time during the development of the program. When writing a test for a class, the programmer can further develop tests for other sections of the program. Furthermore, thorough unit testing encourages the programmer to write clean code, which will help to reduce the chances of bugs and improve the readability of the code.

Task 1(c)

Name, with reason, the type of testing that can guarantee thread-safety of a code fragment. Explain why thread-safety might be important for testing.

There is not one specific type of testing that can guarantee thread-safety in Java, as it's not possible to prove thread safety in the first place, however there are a number of ways to detect some bugs. The issues that can be caused by a lack of thread-safety are extremely difficult to identify and address, and therefore are rare and hard to reproduce. This means that it is extremely important to implement tests that check for thread safety, as any bugs should be solved wherever possible.

In order to help test thread-safety, the programmer must have a thorough understanding of the memory that is used, in order to write tests that can better check for thread safety. To test if a method is thread-safe, the programmer could call the method in parallel from multiple different threads. This must be done for all potential threads in the code fragment, to check if the output is still correct.

Task 2

Consider the following Java code.

```
class Product {
    private double price;
    private String description;

    // Constructors, getters and setters are omitted
}
```

Task 2(a)

Write an implementation of the method boolean equals(Object obj) for class Product that checks the equality of both fields and complies with the standard requirements.

```
public boolean equals(Object obj) {
      return description.equals(price);
}
```

Task 2(b)

Write an implementation of the method int hashCode()for class Product that complies with the standard requirements.

```
@Override
public int hashCode() {
   return Object.hash(price, description);
}
```

Task 2(c)

Discuss the idea behind the Java class BitSet. Briefly describe the situations when it might be the preferred option. Describe other possible ways of representing the same data.

The BitSet class is a form of bit vector that is used to create an array that contains bit values, which will expand in size where necessary. Each bit contained in the array represents a Boolean value, which is used to indicate whether or not an element is present within the set.

The BitSet class is usually only used where thread-safety is not required, as the elements in the set are compact, making bit-wise operations much more efficient. Both HashSets and TreeSets are ways of representing data. TreeSets will ensure that the order remains consistent, whereas HashSets do not.

Task 3(a)

Consider the following Java code.

```
class A implements Runnable {
    static AtomicInteger currentlyRunningCounter
                                = new AtomicInteger();

    public void run() {
        currentlyRunningCounter.incrementAndGet();
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {}
        currentlyRunningCounter.decrementAndGet();
    }

    public static void main(String[] argv) throws Exception {
        Thread t1 = new Thread(new A());
        Thread t2 = new Thread(new A());
        t1.start(); t2.start();
        System.out.println(currentlyRunningCounter);
        t1.join(); t2.join();
    }
}
```

State the output of this program. If the answer is uncertain, give the bounds. Justify your answer.

The output will be **0** because the master thread will not wait for the other two threads to start/finish.

There is a possibility that if **t1** is fast enough, it will be able to increment before the master thread has been fully executed.

Task 3(b)

Write a thread-safe implementation of a Java function

```
void enqueue(List<Object> list, Object element)
```

that adds the object element to the given list. Specifically, it should be safe to use this function with the same list object in several threads at the same time.

```
void enqueue (List<Object> list, Object element) {
        list = Collections.synchronizedList(list);
        list.add(element);
}
```

Task 3(c)

Describe how the state of thread t changes throughout the execution of the following Java program.

```
class A implements Runnable {
    public void run() {
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {}
    }

    public static void main(String[] argv) throws Exception {
        Thread t = new Thread(new A());
        t.start();
    }
}
```

After a thread has been created, it will go from one state to another in its life cycle, before it is terminated by either the program or the programmer. In the example, the thread **t** is in the new state.

When the **t.start()** method is called on thread **t**, the thread scheduler will move it to a Runnable state.

In that state, the thread **t**, then move to Waiting state for a second, before it moves to Terminate state.

Task 4(a)

Name the two key differences between generics in C# and Java. Discuss possible advantages and disadvantages of the two approaches.

Although Java does not support generics at runtime, C# does contain support for value types, furthermore generics in C# do not require boxing/unboxing.

In Java, the program cannot use primitive type parameters, and therefore has to use wrappers instead, this is due to the fact that generics in Java are language only and can only be implemented within the compiler.

Task 4(b)

The following Java code contains a mistake.

```
List<? extends Integer> x = new ArrayList<Integer>();
x.add(1);
List<? super Integer> y = new ArrayList<Integer>();
y.add(1);
```

State with reason which of the lines will cause a compile-time error.

The second line '**x.add(1);**' will cause a compile-time error as an **int** can't be converted to **Integer**.

Task 4(c)

Describe the concept of array covariance. Explain why it was introduced and what negative consequences it has.

Array covariance allows for programmers to assign an array of a subtype of a class to a variable that has a type of an array of the base type. Both Java and C# will treat an array type as a covariant. When both languages did not support generics, all arrays were invariant.

Array covariance negatively impacts performance, as each time a write operation is carried out on an array, additional runtime checks are performed, which may cause some runtime errors. This may also cause an additional issue with type-safety, when overwriting elements in given arrays.

Task 5(a)

Describe the mechanism used to direct client TCP connections to a specific server application running on the server machine.

In order to direct client TCP connections to a specific server application running on a server machine, one could make use of a socket, which is a is a bi-directional connection that is formed between two programs, on top of a TCP connection.

Sockets are supported across different programming languages and on different platforms. In Java, sockets are associated with input / output streams. The different sockets connected to one server all have the same server address and port number, but also have different client IP addresses / port numbers, therefore the server can distinguish different connections from one client by different port numbers.

Task 5(b)

Discuss why a text-based protocol might be better suited for a service meant to be used by a general audience (e.g. developers in different organisations) such as a web server.

A text-based protocol might be better suited for a service meant to be used by a general audience because the representation is in human-readable format, as opposed to another format such as binary. In the context of developers working in different organisations, a text-based protocol will be much easier to understand, and therefore can be experimented with ease.

For example, with a text-based protocol being used for a web server, it would be possible to split the client and web server processes, which would make it much easier to understand and update. Each request can also be broken down into a single line of text, which makes it easier to identify the process that is occurring at each time. An example of this could be when an error occurs, the error message can simply replace the text that was to be included, making it much easier to identify.

Task 5(c)

Give any two weaknesses of Java serialisation. What possible problems could these cause?

One weakness of Java Serialisation is dealing with different versions of a class, which can cause a problem for long-term persistence. If the attributes of a class are added, deleted, or modified, then problems can occur when attempting to read the previously saved objects in the class.

Another weakness of Java Serialisation is that it isn't easily compatible with other programming frameworks. This means that serialisation-based protocols will often have compatibility issues when working with each other

Task 6(a)

State the output of the following Java code.

```
IntStream stream = IntStream
        .iterate(2, i -> i + 1)
        .filter(i -> IntStream.range(2, i)
                .filter(j -> (i % j == 0))
                .findAny()
                .isEmpty());
stream.limit(5).forEach(v -> System.out.println(v));
```

The prime numbers are output by the Java code:

“““
2
3
5
7
11
”””

Task 6(b)

Intermediate stream operations in Java are lazy. Explain what this means.

Intermediate stream operations in Java are lazy, this is due to the fact that the operation is not evaluated until the terminal output is complete.

In each intermediate operation, a pipeline containing the new stream is created, this then stores the provided function, and returns the new stream.

The pipeline is responsible for the new stream with the operation. When the terminal operation is invoked, the stream will start, and all of the related functions are perform sequentially.

Task 6(c)

Describe the stream produced by the following statement.

```
Stream.generate(new Random()::nextInt)
                        .filter(v -> v % 2 == 0)
```

In the initial state, the stream is meant to produce an infinite random number which is then filtered out by even numbers.

However, in this implementation the function is considered to be lazy. This is because the **generate()** method does not actually generate an infinite random number, as it will only be generated as needed.