Question 1. [5 MARKS]

Part (a) [2 MARKS]

The following code will not compile:

```
public class P {
    private int num;
    P(int n) {
        this.num = n;
    }
}
```

```
public class Q extends P {
    private int i;
    public Q() {
        i = 0;
    }
}
```

Add a single line to class `Q` that will allow the code to compile. It doesn't matter what the code does.

Solution: Class `P` doesn't have a no-arg constructor. Java won't define one for us, since it only does that if there are no constructors at all. In class `Q`'s constructor, the first line does not call `super`, so Java will insert a call to `super()`. But there is no no-arg constructor in the parent class. We must call `super` with some `int` to avoid this.

```
public class Q extends P {

    private int i;

    public Q() {
        super(3);   // New line of code
        i = 0;
    }
}
```

Part (b) [1 MARK]

Suppose we have defined this class:

```
class OddballException extends RuntimeException { . . . }
```

Suppose we have a method called `someCalculation`, and that it calls a helper method that `throws OddballException`. Which of the following is true? Circle one.

1. `someCalculation` must put that call inside a `try-catch` clause.
2. `someCalculation` must declare that it may throw a `RuntimeException`.
3. `someCalculation` must either put that call inside a `try-catch` clause or declare that it may throw a `RuntimeException`.
4. Trick question! You can't extend `RuntimeException`.
5. None of the above.

Part (c) [1 MARK]

What is the difference between a static nested class and an inner class? Circle one.

1. An inner class is defined inside another class and a static nested class is defined inside a method.
2. An inner class is defined inside a method and a static nested class is defined inside another class.

3. Both are defined inside another class, but only static nested classes are static.
4. Both are defined inside a method, but only static nested classes are static.
5. There is no difference; these terms are synonyms.

Part (d) [1 MARK]

Suppose you are testing a class `Something` that includes a public method `alpha` and a private method `beta`. Suppose that `alpha` calls `beta`. Without changing anything in the class, you can only test `beta` indirectly. If you could change the accessibility modifiers of one or both methods, what would you recommend in order to make it possible to test `beta` directly? Circle one.

1. Make `beta` public.
2. Make `beta` public during testing and change it back to private afterwards.
3. Use the keyword `protected`, so that `beta` will be package private.
4. Use no accessibility keyword, so that `beta` will be package private.

Question 2. [7 MARKS]

Consider the following outline of code:

```
interface Blah { ... }
class B implements Blah { ... }
abstract class Clem { ... }
class C extends Clem { ... }
class Dreft { ... }
class D extends Dreft { ... }
```

Part (a) [3 MARKS]

For each line of code below, circle the right answer to indicate whether or not it compiles.

`Blah x = new B();`	[compiles]	does not compile
`B y = new Blah();`	compiles	[does not compile]
`B z = (B) new Blah();`	compiles	[does not compile]

Part (b) [1 MARK]

Must `B` implement all the unimplemented methods of `Blah`?

1. [Yes.]
2. No. If it doesn't, we must not make an instance of `B`.
3. No. If it doesn't, we can make an instance of `B`, but we must not call the unimplemented methods.
4. `Blah` cannot have any unimplemented methods.

Part (c) [1 MARK]

Can `Blah` implement any methods?

1. It may declare static methods but not instance methods.
2. It may declare instance methods but not static methods.
3. It may declare both static methods and instance methods.
4. [No. It cannot declare static methods or instance methods.]

Part (d) [1 MARK]

Can `Clem` declare any variables?

1. It may declare static variables but not instance variables.
2. It may declare instance variables but not static variables.
3. [It may declare both static variables and instance variables.]
4. No. It cannot declare static variables or instance variables.

Part (e) [1 MARK]

Must `D` implement all abstract methods of `Dreft`?

1. It may declare static variables but not instance variables.
2. It may declare instance variables but not static variables.
3. It may declare both static variables and instance variables.
4. No. It cannot declare static variables or instance variables.

Question 3. [6 MARKS]

Here is some code from two packages called `exam` and `other`.

```
package exam;
public class Thing {

    private int b;
    int c;
    protected int d;

    public void reveal(Thing other) {
        other.b = 66;   // ok
        other.c = 77;   // ok
        other.d = 88;   // ok
    }
}
```

```
package exam;
public class SpecialThing extends Thing {

    public void tryAccess() {
        b = 1;   // not ok
        c = 2;   // ok
        d = 3;   // ok
    }
}
```

```
package exam;
public class AccessDemo {

    public static void main(String[] args) {
        Thing what = new Thing();
        what.b = 11;   // not ok
        what.c = 12;   // ok
        what.d = 13;   // ok
    }
}
```

```
package other;
import exam.Thing;

public class Whatchamacallit {

    public static void tryAccess() {
        Thing huh = new Thing();
        huh.b = 11;   // not ok
        huh.c = 12;   // not ok
        huh.d = 13;   // not ok
    }
}
```

Circle every assignment statement that will not compile and put a checkmark beside every assignment statement that will compile.

Question 4. [7 MARKS]

Your answers to these questions will be marked both on content and on the quality of your writing. Provide a suitable level of detail (given the space available), and pay attention to clarity, conciseness, spelling and grammar. Also remember that I have to be able to read your writing.

How did your team choose items from your product backlog to go on your sprint backlog at each scrum meeting?

Was this strategy effective? Explain why or why not.

How would you recommend changing your strategy if you were to tackle a significantly larger and more complex project in a new team of four?

Question 5. [12 MARKS]

Below is the beginning of a very simple table class. Add to it whatever is necessary to ensure that it does implement `Iterable` as it claims.

Assume that we want to iterate over the individual cells of the table (as opposed iterating over the rows or over the columns), and that we want to go in row-major order. In other words, the second entry we should visit is `contents[0][1]`, not `contents[1][0]`.

```
public class Table implements Iterable {

    /**
     * The contents of this Table.
     */
    private Object[][] contents;

    // Constructors, setters, getters and other table methods omitted.
```

Solution:

```
    @Override
    public Iterator iterator() {
        return new Table.TableIterator();
    }

    /**
     * An Iterator for AddressBook Contacts.
     */
    private class TableIterator implements Iterator {

        /**
         * The row and column of the next cell to return.
         */
        int currentRow;
        int currentColumn;

        public TableIterator() {
            currentRow = 0;
            currentColumn = 0;
        }

        /**
         * Return whether there is another cell to return.
         *
         * @return whether there is another TableCell to return.
         */
        @Override
        public boolean hasNext() {
```

```
            return (currentRow < contents.length &&
                    currentColumn < contents[0].length);
        }

        /**
         * Return the next TableCell.
         *
         * @return the next TableCell.
         */
        @Override
        public Object next() {
            Object res = contents[currentRow][currentColumn];
            if (currentColumn < contents[0].length - 1) {
                currentColumn += 1;
            } else {
                currentRow += 1;
                currentColumn = 0;
            }
            return res;
        }

        /**
         * Remove the TableCell just returned. Unsupported.
         */
        @Override
        public void remove() {
            throw new UnsupportedOperationException("Not supported.");
        }
    }
}
```

Question 6. [12 MARKS]

Twitter is a social networking website where users post short messages called tweets. Posting a message is called tweeting. If user a chooses to "follow" user b, it means that a finds out about b's tweets.

For this question, you will define an `Account` class for keeping track of information about an individual Twitter account. An `AccountList` class, for keeping track of all Twitter accounts, has already been written. The following code demonstrates what these classes must be able to do:

```
public class Twitter {

    public static void main(String[] args) {

        try {
            // Make an account list and create some accounts to go in it.
            AccountList accounts = new AccountList();
            Account a1 = accounts.createAccount("dianelynn");
            Account a2 = accounts.createAccount("barack");
            Account a3 = accounts.createAccount("Oprah");

            // Record the fact that a2 tweeted "Hello world".
            a2.tweet("Hello world.");
            // Record the fact that "dianelynn" now follows "barack".
            // From now on, she will find out about his tweets.
            accounts.recordFollows("dianelynn", "barack");

            // More twitter actions.
            a2.tweet("I love rutabagas!");
            a1.tweet("Me too!!");
            accounts.recordFollows("barack", "Oprah");
            a2.tweet("Totally.");
            a3.tweet("Are you kidding @barack?");

        } catch (UsernameUnavailableException ex) {
            System.out.println("Username already taken");
        } catch (NoSuchUsernameException ex) {
            System.out.println("Unrecognized username");
        }
    }
}
```

With the two classes properly defined, the above code should produce the following output:

```
dianelynn (0 tweets) found out that barack (2 tweets) tweeted 'I love rutabagas!'
dianelynn (1 tweets) found out that barack (3 tweets) tweeted 'Totally.'
barack (3 tweets) found out that Oprah (1 tweets) tweeted 'Are you kidding @barack?'
```

Note: the existing code brings up a number of interesting design issues, but for this question, you don't need to be concerned about that.

Assume that classes `UsernameUnavailableException` and `NoSuchUsernameException` have been appropriately defined. On the next page, you will find class `AccountList`. On the two pages after that, write the one missing class: `Account`. You must use the Observer design pattern. Hints:

- Design your code so that an `Account` object can be observed and also can observe other `Account` objects.
- In class `Account`, store only the username and number of tweets made by that user. For the purposes of this question, you don't need to store the actual tweets.

Implement only the methods called in the existing code and anything needed to make them work as described by the output above. For example, you do not need to write any getters of setters.

To earn credit for your answer, you *must* use the observer pattern as described above. Some of the marks will be for good coding style. You do not need to write any Javadoc.

```
public class AccountList {
    private HashMap<String, Account> list;

    public AccountList() {
        this.list = new HashMap<String, Account>();
    }

    /**
     * Creates a new account with the specified username and remembers it in
     * this AccountList.
     *
     * @param username the username for the new account.
     * @return the new account.
     * @throws UsernameUnavailableException if the specified username has
     * already been used for another account.
     */
    public Account createAccount(String username)
            throws UsernameUnavailableException {

        if (list.containsKey(username)) {
            throw new UsernameUnavailableException();
        } else {
            Account a = new Account(username);
            list.put(username, a);
            return a;
        }
    }

    /**
     * Records the fact that s1 follows s2.
     *
     * @param s1 the username of the person who follows s2.
     * @param s2 the username of the person followed by s1.
     * @throws NoSuchUsernameException if either s2 or s2 is not a username for
     * an existing account.
     */
    public void recordFollows(String s1, String s2)
            throws NoSuchUsernameException {

        Account a1 = this.list.get(s1);
        Account a2 = this.list.get(s2);
        if (a1 == null || a2 == null) {
            throw new NoSuchUsernameException();
        } else {
            a1.follows(a2);
        }
    }
}
```

Solution:

```
public class Account extends Observable implements Observer {

    private String account;
    private int numTweets;
    private ArrayList<String> tweets;

    public Account(String who) {
        this.account = who;
        this.numTweets = 0;
    }

    public String toString() {
        String result = String.format("%s (%s tweets)",
                this.account, this.numTweets);
        return new String(result);
    }

    /**
     * Record the fact that this member follows other.
     * @param other the member that this member follows.
     */
    public void follows(Account other) {
        other.addObserver(this);
    }

    public void tweet(String message) {
        this.numTweets += 1;
        this.setChanged();
        this.notifyObservers(message);
    }

    @Override
    public void update(Observable o, Object tweet) {
        System.out.println(
                String.format("%s found out that %s tweeted '%s'",
                this, o, tweet));
    }

}
```

Question 7. [13 MARKS]

Part (a) [2 MARKS]

Write all the strings that match this regular expression: `[ho]?ho?[ho]`

Part (b) [6 MARKS]

Write a regular expression for each of the following languages. Write your regular expressions in the mathematical sense, not as Java `Strings`.

All strings of 0s and 1s that have at least 3 leading 0s.

All binary strings of length zero or more in which every odd position contains the digit 1. (Treat the first character of the string as being in position 1.)

All strings consisting of any number of a's and then either a single 0 if the number of a's was even, or a single 1 if the number of a's was odd.

Part (c) [4 MARKS]

Complete the method below. You may assume that all appropriate imports have been done. For efficiency, you must compile the pattern once and reuse it. Note: there is a generous amount of space for your answer — don't feel you need to fill it all.

```
/**
 * Return the number of strings in data that match regex.
 *
 * @param regex a regular expression
 * @param data the strings to be matched against regex
 * @return the number of strings in data that match regex
 */
public static int numMatches(String regex, ArrayList<String> data) {



    int count = 0;



    for (                                                ) {









    }

    return count;

}
```

Solution:

```
public static int numMatches(String regex, ArrayList<String> data) {

    Pattern p = Pattern.compile(regex);

    int count = 0;
    for (String s : data) {
        Matcher m = p.matcher(s);
        if (m.matches()) {
            count += 1;
        }
    }
```

```
    return count;
}
```

Part (d) [1 MARK]

While of the following is true? Circle one.

1. There are languages you can describe with BNF (Backus-Naur Form) that you cannot describe with regular
2. There are languages you can describe with regular expressions that you cannot describe with BNF.
3. The set of languages you can describe with regular expressions is exactly the same as the set of languages you can describe with BNF.
4. Trick question! You can't describe a language with either one of these notations.

Question 8. [11 MARKS]

Part (a) [7 MARKS]

Complete the following method. If it uses any methods that may throw an exception, have method `reportGettersAndSetters` simply pass the exception along rather than catch and handle it.

Note: It would be better style for `reportGettersAndSetters` to throw more specific types of exceptions. They are lumped together here as merely `Exception`s just for simplicity.

Solutions:

```
/**
 * Reports on the number of setters and getters in the class named c.
 *
 * @param c the name of the class to be analyzed.
 * @return an array of 2 elements which stores at index 0 the number of
 * methods whose names begin with "set", and stores at index 1 the number of
 * methods whose names begin with "get".
 * @throws [details omitted]
 */
public static int[] reportGettersAndSetters(String c) throws Exception {

    int[] answer = new int[2];
    answer[0] = 0;
    answer[1] = 0;

    Class thisClass = Class.forName(c);

    Method[] methods = thisClass.getDeclaredMethods();
    for (Method m : methods) {
        if (m.getName().startsWith("set")) {
            answer[0] += 1;
        } else if (m.getName().startsWith("get")) {
            answer[1] += 1;
        }
    }
    return answer;
}
```

Part (b) [4 MARKS]

Write a main method (to go in the same class) that will use `reportGettersAndSetters` to find out about class `HashMap`. If any exceptions are thrown, have it print "Sorry" rather than letting the user see any exceptions. Make it produce output like this:

```
Getters: 0
Setters: 3
```

(but of course with whatever numbers are correct).

Solutions:

```
public static void main(String[] args) {
    try {
        int[] report = reportGettersAndSetters("java.util.HashMap");
        System.out.println(
                String.format("Getters: %s\nSetters: %s",
                report[0], report[1]));
    } catch (Exception ex) {
        System.out.println("Sorry!");
    }
}
```