Student number:	2467273S
Course title:	COMPSCI2001 Java Programming 2
Questions answered:	ALL

1.

(a)

```
public class TeamRecord {
	private String teamName;
	private int points;

	public TeamRecord(String team) {
		this.teamName = team;
		this.points = 0;
	}

	public String getTeamName() {
		return this.teamName;
	}

	public int getPoints() {
		return this.points;
	}
}
```

(b)

```
public void updateScore(int homeScore, int oppScore) {
		if(homeScore == oppScore) {
			this.points += 1;
		}
		else if(homeScore > oppScore) {
			this.points += 3;
		}
	}
```

(c)

Since there is an unchanging number of teams in the league, an array of TeamRecord objects would be appropriate:

```
TeamRecord[] teams = new TeamRecord[12];
```

(d)

```
@Override
	public int compareTo(TeamRecord other) {
		if (this.points != other.getPoints()) {
			return other.getPoints() - this.points;
		} else {
			return this.teamName.compareTo(other.getTeamName());
		}
	}
```

(e)

Add “implements Comparable” with the specific TeamRecord type:

```
public class TeamRecord implements Comparable<TeamRecord>
```

(f)

Assuming java.util.Arrays is imported:

```
Arrays.sort(teams);
```

2.

(a)

(i)

Method signature: setA(int a)

- setA is the method name that describes the action being done
- int a is a parameter that is given as an argument when calling the method

(ii)

Line 19 sets a local variable “a” that is visible to the main method. Line 20 uses a method that changes the class’ field, also called “a”, but this is not what is being referred to in Line 21 because there is an “a” taking precedence in local scope.

(iii)

Line 23 creates a new instance of ExamQ2 (an object) called “ee” with an argument of “5”, which sets “ee”’s field to 5. Then Line 24 calls a method which sets “ee”’s field to that of the original (accessed by a getter) (5) plus 3 (8). Then Line 25 prints out “ee”’s “a” since a getter is used, which is 8 (5+3).

(b)

(i)

Since the behaviour of the methods is not dependent on any instance of the class, but rather they describe general behaviour for *any* instance, they are static and should be declared thus. If this was the case, the main method should call them with “Trims.leftTrim(s)” and “Trims.rightTrim(s)” inside the print statements, but it would compile either way.

(ii)

It would not because the main method is inside the Trims class; thus, private methods would be accessible too since they are within the scope.

(iii)

```
System.out.println(leftTrim(rightTrim(s)));
```

(iv)

A “for” loop would require knowing how many whitespace characters there are before/after the string; however, this is not known, so a “while” loop is the most obvious choice to continue looping until some condition is satisfied.

3.

(a)

When “equals()” is overridden, two objects are considered to be equal according to a different set of rules instead of the default “references are equal for both objects”, which is rarely what programmers care about when comparing two different objects. Thus, when two objects are equal according to different rules, their equality should be represented by their hash codes as well, i.e., “hashCode()” has to be overridden, too. This override is an instance of polymorphism, as it defines more specialised behaviour of equality for classes that are not just Objects (which is the superclass of every class in Java). Polymorphism allows to use “equals()” and “hashCode()” for subclasses of Object in a more specific way than the default methods would allow.

(b)

Advantage:

- **Shorter and non-bloated code** enables/improves legibility, debugging, memory space of the program, etc

Disadvantage:

- The code should be prepared to deal with frequent cases where exceptions could occur as it would then help the client understand what they/their computer is doing wrong, i.e., human errors are frequent and should be easy to resolve.

(c)

Advantage:

- Makes it harder for a programmer to write incorrect code, which would make any program with only checked exceptions very durable and stable.

Disadvantage:

- The code would be **bloated with unnecessary exception handling** that can be solved easily with better programming style/more careful programming. Bloated code would then take up more space, be less legible, harder to debug, etc.