Marking Guidelines

Overall guidelines:

1. Code that does not compile receives a maximum mark of **40%** (8 / 20).
2. Code that completes the requirements procedurally, as opposed to using the relevant object-oriented principles, receives a maximum mark of **40%** (8 / 20).
3. **1** mark is deducted unnecessary or **inefficient** steps, e.g. storing the result of a method in a variable, and then printing that variable, rather than simply printing the method result directly; creating unnecessary objects; or including additional methods that are not in the model solution.
4. Marks are not deducted for additional print statements (e.g. `Round 1 Starts').
5. These guidelines are **not exhaustive**.

Question 1. (A)

Create a class to represent a Superhero. Every Superhero has a name. Every Superhero can also have a strength, but this should be optional. If a Superhero does not have a strength, their default strength should be 10. **(5 marks)**

```
public class Superhero {
	private String name;
	private int strength;
	public Superhero( String name ) {
		this.name = name;
		strength = 10;
	}

	public Superhero( String name, int strength ) {
		this.name = name;
		this.strength = strength;
	}
```

5 marks = Two private variables of type String and Name. Use of two constructors to make the strength optional upon instantiation. Setting a default value in first constructor. The use of `this' is optional.

Mark deduction (until 0):

- Not using a constructor to set the name of a superhero **(-4)**
- A missing field (either name or strength) or a field (either name or strength) that is not assigned a value by a method **(-3)**
- A single constructor, accepting both the name and the strength, that uses a conditional to check if the strength is present or not **(-2)**
- Public fields (fields with no access modifiers are public by default) **(-2)**
- Default strength is not set to 10 in any way **(-2)**
- Use of an additional method that sets the strength after construction **(-1)**
- Setting the default strength at the field, rather than in a constructor **(-1)**
- Additional fields storing unnecessary information (such as a superhero's powerup) **(-1)**
- Using 'this' when it does not differentiate between the parameters and the fields **(-1)**

Question 1. (B)

Every Superhero can receive a powerUp, whereby their strength is increased by a specified amount. **(3 marks)**

```
public void powerUp( int strengthIncrease ) {
      strength += strengthIncrease;
}
```

3 marks = A single method without a return type, that accepts an integer and adds this on to the strength value. Additional logical, checks, such as whether the strength increase is greater than zero, are fine.

Mark deduction (until 0):

- No powerup method **(-3)**
- Replacing the strength rather than increasing it **(-2)**
- Returning something from the method **(-1)**
- Additional parameters **(-1)**

Question 1. (C)

Every Superhero has the ability to fight another Superhero. The result of this fight is the winning Superhero. The winner of a fight is determined by which hero has the highest strength. If two heroes have the same strength, then the opponent wins. (It might be helpful to know that the keyword this returns a copy of the current object.) **(4 marks)**

```
public Superhero fight( Superhero opponent ) {
    if ( strength > opponent.strength ) {
        return this;
    } else {
        return opponent;
    }
}
```

4 marks = A method called fight that accepts one Superhero object, and returns another Superhero object. If the strength in the current object is greater than the strength in the opponent object, return this object as the winner, otherwise return the opponent object.

Mark deduction (until 0):

- No fight (or similarly named) method **(-4)**
- No fight method, but a calculation of who wins in the Fight class (main method) **(-3)**
- Printing the winner in the fight method, not in the Fight class **(-3)**
- Using a method such as 'getStrength' to access an opponent's strength, rather than accessing the fields directly **(-2)**
- Incorrect calculation of the winner **(-2)**
- Returning a String rather than a Superhero from the method **(-1)**
- Unnecessary conditional checks. An elseif statement for the situation in which both strengths are equals is fine **(-1)**
- Using 'this' without it being necessary to differentiate between parameters and fields **(-1)**

Question 1. (D)

When a Superhero is printed, their name appears on the terminal. **(2 marks)**

```
public String toString() {
	return name;
}
```

2 marks = A simple toString() method that returns the name of the Superhero, allowing a Superhero object to become printable.

Mark deduction (until 0):

- Anything other than 'public String toString()' **(-2)**
- Using a 'getName' method, or similar, to return the name, rather than toString() **(-2)**
- Returning extra information, in addition to the name **(-1)**

Question 2. (A)

Note: Adding user input here is **not** expected. If user input has been used, the strength values and the superhero names assigned must be maintained.

Create a Superhero named Iron Man. (1 mark)

```
Superhero ironMan = new Superhero("Iron Man");
```

1 mark = Creation of the Superhero object. If students wish to define the Superhero's name in a separate String variable as a form of constant, and then input this to the constructor, that is fine.

Mark deduction (until 0):

- Not specifying a superhero's name **(-1)**
- Specifying anything other than the first superhero's name (e.g. their strength) **(-1)**
- Not using the first superhero name assigned **(-1)**

Question 2. (B)

Create a Superhero named Captain America. Captain America. has a strength of 17. **(1 mark)**

```
Superhero captainAmerica = new Superhero("Captain America", 50);
```

1 mark = Creation of the second Superhero object. If students wish to define the Superhero's name and strength in separate variables as a form of constant, and then input these to the constructor, that is fine. If a superhero's strength is set using hero2.setStrength (or similar), do not deduct additional marks.

Mark deduction (until 0):

- Not specifying a superhero's name **(-1)**
- Not specifying a superhero's strength in any way **(-1)**
- Not using the second superhero name assigned **(-1)**
- Not using the strength assigned **(-1)**

Question 2. (C)

Make Iron Man fight Captain America. Print the winner to the terminal. (2 marks)

```
System.out.println(ironMan + " fights " + captainAmerica + ", " +
ironMan.fight(captainAmerica) + " wins.");
```

2 marks = One simple call the HeroA.fight(HeroB) to print the winner. Details about who fights who are nice, but not essential.

Mark deduction (until 0):

- No information about the winning superhero **(-2)**
- Calculates the winner of the fight in the Fight class but the result is incorrect **(-2)**
- The result of the fight method is not used to print the result (e.g. the method is void, toString() is not used, or a Superhero or a String are returned, but not used for printing) **(-1)**
- Explicitly calling toString() on an object **(-1)**
- The result of the fight is a String, so the winner of the fight is printed as a String, rather than invoking toString() automatically to print the name inside the winning superhero object **(-1)**
- Prints the results of the fight by getting information from the Superhero class, such as a Superhero's strength, and compares the strength in the Fight class, but the result is correct **(-1)**

Question 2. (D)

Give Iron Man a powerUp of 100. (1 mark)

```
ironMan.powerUp(100);
```

Mark deduction (until 0):

- Not giving the superhero a power **(-1)**
- Passing anything other than a single integer to the powerUp method **(-1)**

Question 2. (E)

Make Iron Man fight Captain America, again. Print the winner to the terminal

```
System.out.println(ironMan + " fights " + captainAmerica + ", " +
ironMan.fight(captainAmerica) + " wins.");
```

This question is not worth any marks.

Award one mark for the presence of comments. Should be at least one comment per method. More is fine.

The marks awarded to you as a result of these guidelines are only provisional. Your actual grade is based upon an assessment of how competently you can answer the questions posed to you by your examiner during your assessment.

These grades are scaled down to a portion of the 14% available for the second assignment.