FIT2099 Sample Questions and Answers

Question 1 – (6 + 6 = 12 marks)

You have joined a team that is working on a fully-featured version of the game similar to the one you worked on in your assignments, complete with a graphical user interface. It is to be released commercially. The plan is to re-use the engine code for similar games in the future.

The game runs reasonably well, but when you look at the engine code you discover that it contains a lot of "cut-and-paste reuse" – that is, repeated similar sequences of code, sometimes with a few very small changes.

a) Why do software engineers consider repeated code to be a bad thing? Write a paragraph or two about the risks of cut-and-paste reuse. (6 marks)

b) Describe how you would go about fixing the code to reduce these risks. (6 marks)

a)

- changing the code in one place means changing code in all others
    - waste time refactoring
- creates implicit dependencies
    - changing one requires changing the others (only if they work the same way)
- adding new features is harder
- makes it harder to read
    - more stuff to read
    - harder to understand
        - is it REALLY the same?
            - Should it be the same
        - Why is it repeated
- Indicates a lack of understanding
    - Did the person writing this really understand what they were doing?

Repeated code is considered a bad thing as it can cause problems while refactoring as well as creating confusion for everyone developing the project.

Repeated code can create implied dependencies on all other instances of repeated code. This is because, if the code is supposed to the same or similar thing, then having to refactor and change the code would likely mean having to change the code at all instances. This can be problematic, as not only does it waste time for the developer to make the change at every location, it also adds the risk of missing a location, or making the change wrong, which could cause unforeseen issues possibly much further down the development process.

Another issue is readability. Clearly, having large chunks of the same code is much harder to understand than one chunk that is organised properly. This wastes time as developers will have attempt to understand and decipher the code in multiple places, as well as having to read the same code multiple times. It also begs the question if the code is really the same, which will have to be analysed at every instance and if it is, why? And if it's not, should they be the same? To find these answers, a very thorough understanding of the code must be developed, which may not always be necessary and is likely a waste of time and energy.

By having repeated code, this would typically mean that the developer lacks an understanding of how the code works, as they likely just copied and pasted other parts and hoped they worked. This will be problematic later, as someone who cannot create the code properly probably won't be able to maintain it, or help others maintain it.

b)

- Put into a Loop
    - If they are immediately after each other
- Put into a Method
    - If they are in the same class
- Create a new superclass
    - If they are over different classes
    - Hint they are all subclass of something that should exist
    - Be careful of typing, they must still abide by their type

In order to fix the code and reduce the risks, the placement of the repeated code must first be looked at.

If the repeated code is being called immediately one after the other, a simple loop will be able to solve the problem. This will allow the loop to always run the correct amount of times, and gives a reason why the code is being repeated

If the code is being repeated throughout the same class, creating a method will improve readability. This allows the code to be more organised, as it now has a method that can be called. This allows for easier delegation and clarity, and developers can easily see why the section of code is being called.

If the code is present throughout multiple class, this hints that they are all subclasses of something that should exist, and thus should be more closely grouped together, and so a new superclass should be created. This will allow for the repeated code to disappear, as well as grouping together classes that should be more closely connected than they previously were.

Question 2 - (4 + 6 = 10 marks)

In mathematics, an integer is a number that is the member of the set {…,-2, -1, 0, 1, 2,…}

A rational number is any number that can be expressed as a fraction p/q, where p and q are integers and q is not 0.

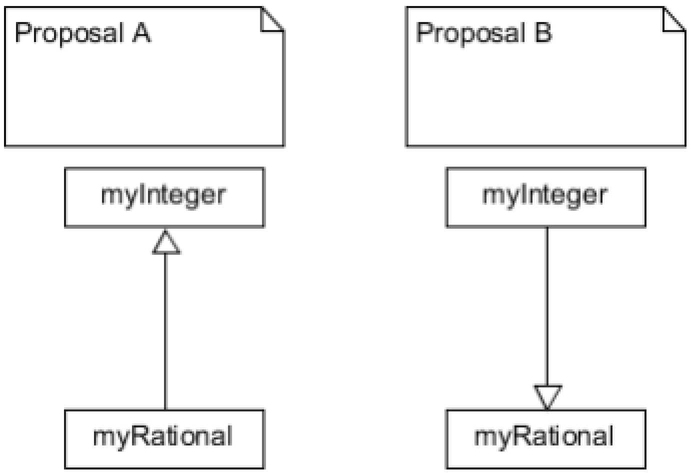
Trivially, any integer x can be unambiguously expressed as a rational number x/1, so any integer is also a rational number.

Imagine you wished to define two classes, myInteger and myRational, that (initially) support the following operations:

• Constructing new objects with an initial value

• Incrementing the value by the value of another myInteger or myRational object.

Consider two class diagrams below that show proposed inheritance relationships between myInteger and myRational



a) In a couple of paragraphs, explain the advantages and disadvantages of both proposal A and proposal B, considering the design principles relating to inheritance discussed in FIT2099. (6 marks)

b) Propose an alternative design that avoids the disadvantages of both proposal A and proposal B. Draw a class diagram to illustrate it, and explain clearly how your proposal would implement the required operation. (6 marks)

a)

- Proposal A
    - Liskov Substitution Principle
        - A rational is not a special type of integer
        - Integer + integer = rational (should be integer)
            - Can give a rational where an integer was expected
- Proposal B
    - Computing operations issues

        - Creating an myInteger will have a stricter constructor than myRational

The issue with proposal A is that it violates the Liskov Substitution Principle, which states that for class S to be a true subtype of class T then S must conform T, where an object of class S can be provided where an object of class T is expected and correctness is still preserved. Since a rational is not a special type of integer, and integer + integer = rational, when it should be an integer, and thus a rational can be given where an integer is expected, when it should not be able to, thus the principle has failed.

Proposal B does not have the problem proposal A has, however, it may have some computing operations issues, as myInteger will have stricter constructor than myRational.

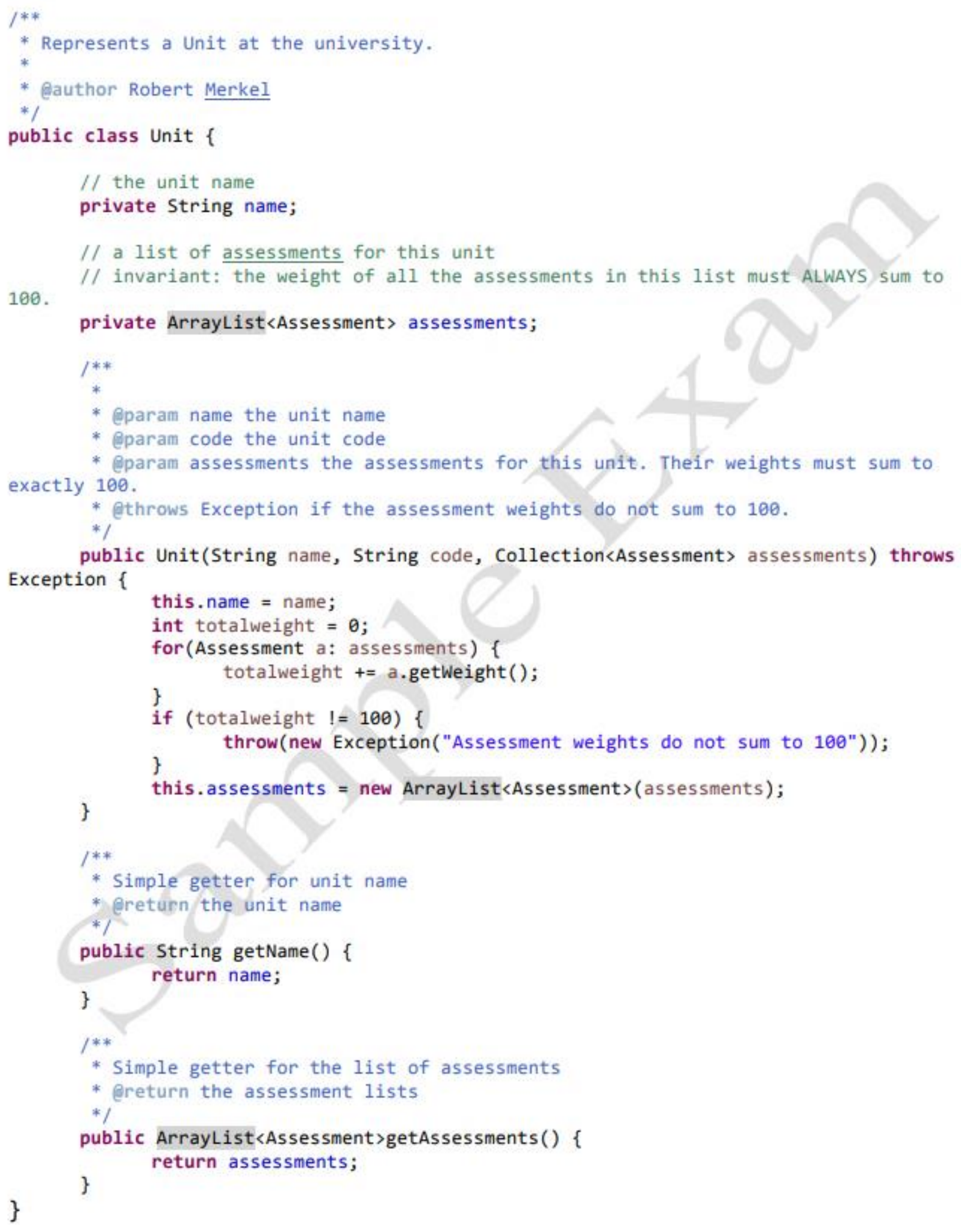
b)

- Create a superclass of number
    - Have integer and rational be a subclasses on myNumber
    - Won't be great, as myNumber will be quite empty (possible interface)
- Not many good solutions

One of very few good solutions to this problem is to create a myNumber superclass and have myInteger and myRational as subclasses of this. This will eliminate the typing and constructor problems that the other proposals had. However, myNumber would be quite empty and may be more beneficial as an interface instead.

Question 3 - (4 + 5 + 6 = 15 marks)

Consider the following Java source code implementing a Unit class for a system similar to JavaUnivesity from your labs:



```java
/**
 * Represents a Unit at the university.
 *
 * @author Robert Merkel
 */
public class Unit {

	// the unit name
	private String name;

	// a list of assessments for this unit
	// invariant: the weight of all the assessments in this list must ALWAYS sum to
100.
	private ArrayList<Assessment> assessments;

	/**
	 *
	 * @param name the unit name
	 * @param code the unit code
	 * @param assessments the assessments for this unit. Their weights must sum to
exactly 100.
	 * @throws Exception if the assessment weights do not sum to 100.
	 */
	public Unit(String name, String code, Collection<Assessment> assessments) throws
Exception {
		this.name = name;
		int totalweight = 0;
		for(Assessment a: assessments) {
			totalweight += a.getWeight();
		}
		if (totalweight != 100) {
			throw(new Exception("Assessment weights do not sum to 100"));
		}
		this.assessments = new ArrayList<Assessment>(assessments);
	}

	/**
	 * Simple getter for unit name
	 * @return the unit name
	 */
	public String getName() {
		return name;
	}

	/**
	 * Simple getter for the list of assessments
	 * @return the assessment lists
	 */
	public ArrayList<Assessment>getAssessments() {
		return assessments;
	}
}
```

a) The class and the class methods both have Javadoc comments. However, the attributes do not – despite the fact that you can annotate attributes with Javadoc. In your own words, briefly explain the reason for using Javadoc comments in your code. Does it matter much if the comments for the attributes in this class do not use Javadoc? Why or why not? (4 marks)

b) There is a potential design problem with getAssessments(). Explain what is wrong, and write an alternative version of getAssessments() that does not have this problem. (7 marks)

c) The Java language specification states that if a method throws a checked exception (that is, any exception that is not an instance of RunTimeException or a subclass), that method must either:
• catch and handle them within its body; or
• declare in its signature that it can throw them back to the caller (e.g. as the constructor for Unit does)

Not all languages that support exception handling have this requirement. In Python, for instance, any method can raise any exception it chooses, and there is no requirement for the method signature to indicate this.

In about a paragraph, explain one advantage and one disadvantage of each approach. (6 marks)

a)

- Allow for easier readability and understanding
    - Useful for both people working on the current code and using it as a library
    - Generates HTML
    - Easy access and information to all parts of the code
    - Saves time and effort for developers
        - Can understand the code much faster
- Attributes do not have Javadoc as they are private
    - Private methods should not have Javadoc
        - Should not be known about outside the class
        - Should not be modified or called outside the class
    - Does not matter in this case

b)

- Privacy leak
    - getAssessments() returns assessments, of mutable type ArrayList, meaning a reference will be returned
        - the reference will be the same as the private attribute assessment reference
            - changing this reference will change the private attribute
    - getAssessments() should return a copy of the ArrayList, rather than the reference
        - this can be done by cloning the ArrayList and return the new clone

c)

- Advantage: Obvious to programmer that it will throw an exception
    - Will know that there will be unacceptable arguments
- Disadvantage: Changing a method to now throw an exception or to no longer throw an exception:
    - Will require to go up the chain until all problems are exceptions are handled (wastes time)

Question 4: 2 + 2 + 4 + 4 = 12 marks

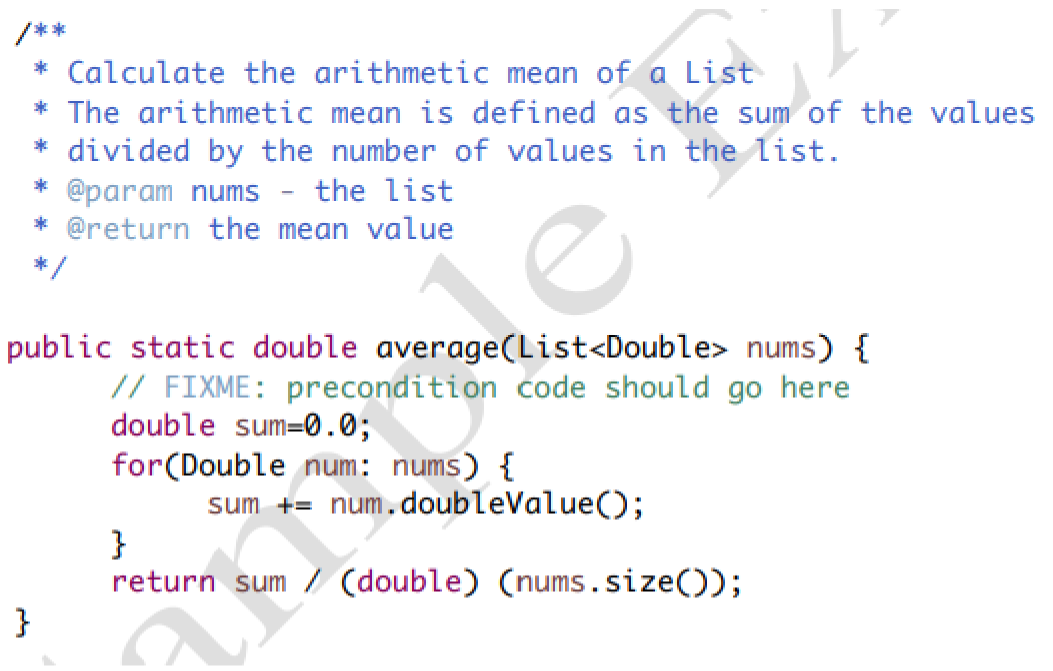
Design By Contract is an approach to software design that is supported directly in some languages, and which can be applied in others, such as Java, through the use of extensions or other language features. In Design By Contract, the designer of a class tells the clients of the class what the class does and what it needs via the specification of the class.

(a) Name two features of Java that allow a developer to write executable preconditions, and explain how these features can be used for this purpose. (2 marks)

(b) Why is it a good idea for a method not to try to cope with a violation of its preconditions, but simply to report the problem to the client that called it? (2 marks)

(c) Give an example of a method for which it would be much easier to write a postcondition than to write the body of the method. Explain when and how writing such a postcondition would aid in software development. (4 marks)

(d) Here is a simple Java method, in which the programmer would like to use a precondition. Choose a useful precondition for this method, explain what it is (in English), and write Java code to enforce the precondition (using standard Java, without using CoFoJa extensions). (4 marks)



```
/**
 * Calculate the arithmetic mean of a List
 * The arithmetic mean is defined as the sum of the values
 * divided by the number of values in the list.
 * @param nums - the list
 * @return the mean value
 */

public static double average(List<Double> nums) {
       // FIXME: precondition code should go here
       double sum=0.0;
       for(Double num: nums) {
              sum += num.doubleValue();
       }
       return sum / (double) (nums.size());
}
```

a)

- Assertions
    - Allow for the precondition to be checked upon entering the function, and post condition upon exiting. Assertions check if a statement is true, and can thus be used to ensure that the data being received and returned is correct
- Exceptions
    - Exceptions allow for a system to try something and fail. If it fails, it can be caught within the system, or be thrown back up the chain until it is handled. This can check for preconditions and postconditions by trying something to check for its conditions, and then throwing an exception back if it is not handled properly

        - This one is preferred for precondition specially (java policy document)
            - Assertions can be turned off
            - Preconditions violation is usually because of a bug

b)

- Cannot assume what is expected
    - It's the client's fault
        - They should fix it
        - They're the only ones to know how to fix it
    - If we try to assume, we could be ignoring a bug given early
        - E.g. find the root of a given negative, we shouldn't find the modulus
            - This likely due to a calculation error somewhere else in the code
- Fail fast
    - If given something that is unexpected, throw an error earlier. That way bugs can be found a fixed earlier

c)

- Finding the root of a number
    - May be difficult to calculate
    - To check, just square the answer, and it should be the original number
- Sorting algorithm
    - May be hard to sort the list
    - Iterate over the list to check
- Ensures all inputs, calculations and return values are correct
    - A test built into the method

d)

- Ensure the list has items in it
    - The division of will throw a divide by zero error
- Assert (nums.size()>0);

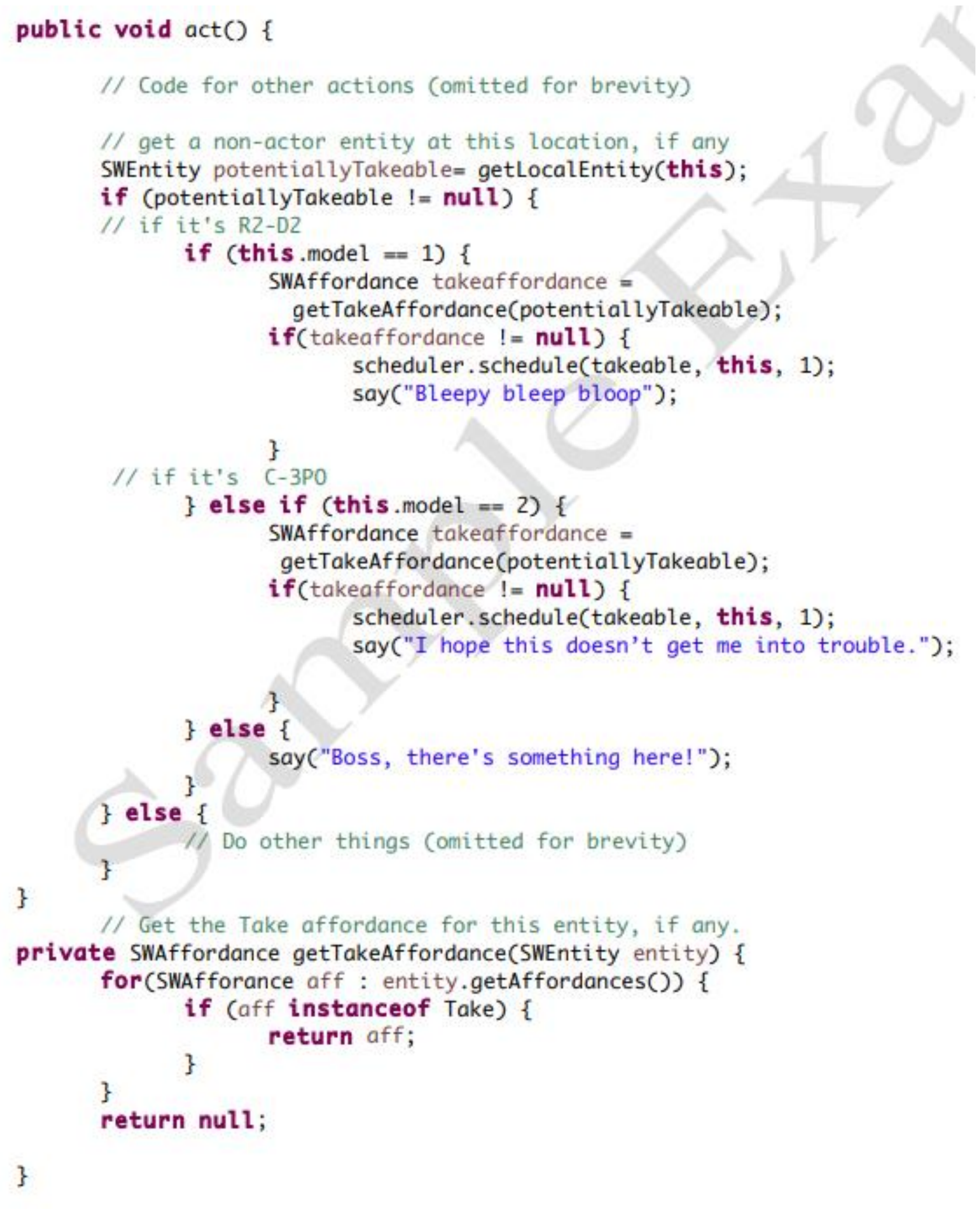
Question 5: 10 marks

Below is some code from the act() method of a Droid in Star Wars game using the same rogue-like game engine as in your assignments.

Review this code for maintainability and extensibility.

For each design problem you identify:
a) Describe the problem clearly.
b) In a sentence or two, explain potential negative consequences of the problem.
c) Outline a potential fix for each design problem. You do not need to write actual code, but explain your fix in enough detail to make clear how your fix would address the problem. Write up to a few sentences.

Hint: consider “code smells”…



```
public void act() {

        // Code for other actions (omitted for brevity)

        // get a non-actor entity at this location, if any
        SWEntity potentiallyTakeable= getLocalEntity(this);
        if (potentiallyTakeable != null) {
        // if it's R2-D2
                if (this.model == 1) {
                        SWAffordance takeaffordance =
                          getTakeAffordance(potentiallyTakeable);
                        if(takeaffordance != null) {
                                scheduler.schedule(takeable, this, 1);
                                say("Bleepy bleep bloop");

                        }
         // if it's  C-3PO
                } else if (this.model == 2) {
                        SWAffordance takeaffordance =
                         getTakeAffordance(potentiallyTakeable);
                        if(takeaffordance != null) {
                                scheduler.schedule(takeable, this, 1);
                                say("I hope this doesn’t get me into trouble.");

                        }
                } else {
                        say("Boss, there's something here!");
                }
        } else {
                // Do other things (omitted for brevity)
        }
}
        // Get the Take affordance for this entity, if any.
private SWAffordance getTakeAffordance(SWEntity entity) {
        for(SWAfforance aff : entity.getAffordances()) {
                if (aff instanceof Take) {
                        return aff;
                }
        }
        return null;

}
```

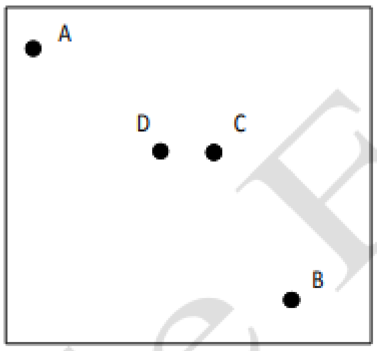
- The two if conditions
    - Speculative Generality – add methods for every special case
    - Adding new special cases is hard
    - Is messy and hard to understand
    - Repeated code – DRY principle
    - Removing old special cases can cause bugs
    - Create a new subclass specifically for the special cases
- Checking if aff instance of Take
    - Dependency on Take
        - Changed to Take will affect this code
    - Make the entity have a Boolean attribute isTakeable that knows if it can be taken
- Magic numbers
    - Can be hard to figure out a meaning behind them
    - Should be changed to a constant or enum
    - This will ensure that the number cannot be changed

Question 6: 5 + 3 + 5 + 7 = 20 marks

You're working on a system to monitor pairs of primary school students working on a geography assignment on the grounds of their school, which is too large for a teacher to keep all the students in sight at all times.

The students are supposed to work on the assignment in assigned pairs. Each student carries a smartphone with an app that periodically reports the position of that student to a monitoring system. The positions are reported as two numbers, x and y, which represent the offset in an easterly and northerly direction, respectively, in metres from an origin point in the center of the school. For instance, a student at the point (20.5, -5.3), would be located at a point 20.5 metres east and 5.3 metres south of the origin point.

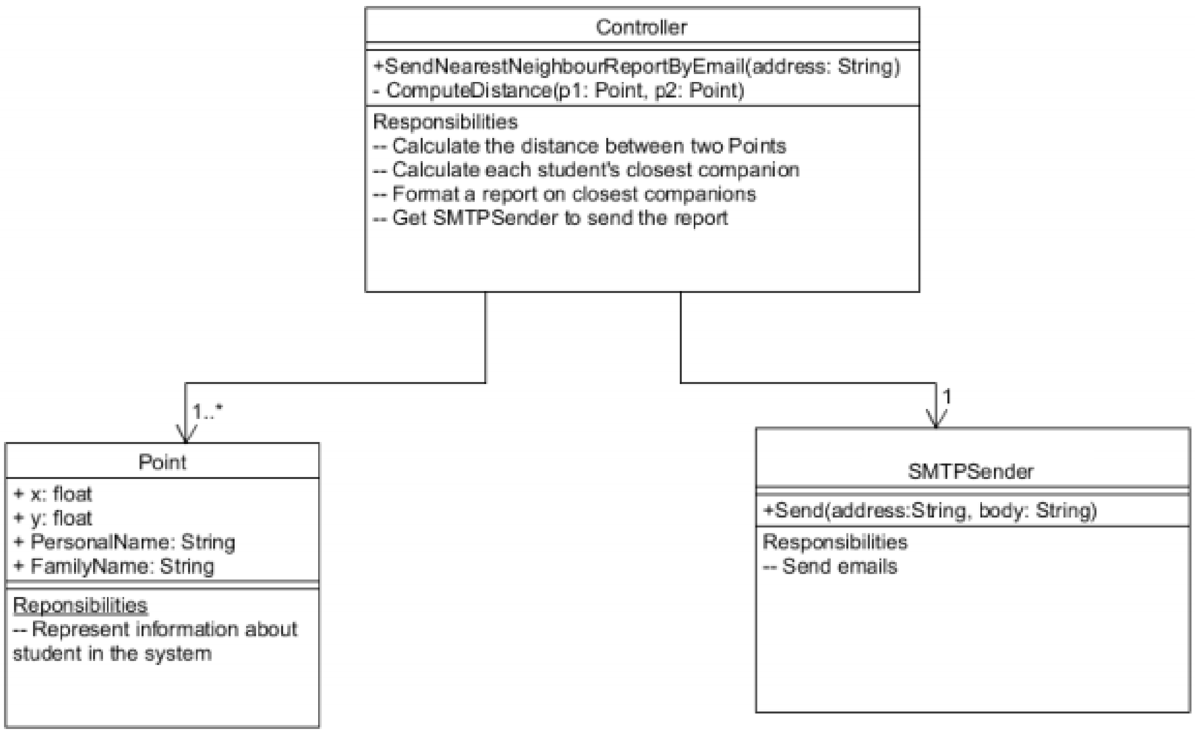
To help teachers verify that they are working in their pairs, the system should regularly email the teacher a report listing each student and the student closest to them at that moment. For instance, if there are four students, A, B, C, and D, located as follows:



The emailed report should read:

A is closest to D
B is closest to C
C is closest to D
D is closest to C

You and a colleague are working on a design for the part of the monitoring system that emails the report. They send you the following class diagram, annotated with responsibilities for classes.

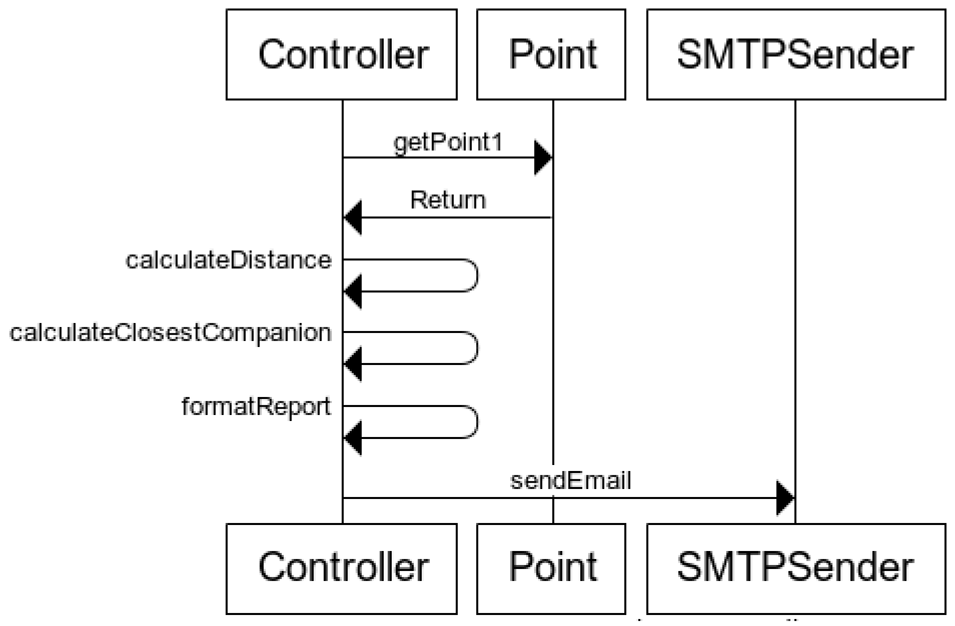

Controller
+SendNearestNeighbourReportByEmail(address: String)
- ComputeDistance(p1: Point, p2: Point)
Responsibilities
-- Calculate the distance between two Points
-- Calculate each student's closest companion
-- Format a report on closest companions
-- Get SMTPSender to send the report
1..*
1
Point
+ x: float
+ y: float
+ PersonalName: String
+ FamilyName: String
Reponsibilities
-- Represent information about student in the system
SMTPSender
+Send(address:String, body: String)
Responsibilities
-- Send emails

a) Based on the description and the class diagram above, draw a sequence diagram for the "email nearest neighbour report". (5 marks)

b) How easy would it be to extend this system to support sending the report by other means? In a few sentences, explain the features of the design that make it easy or hard, and why they would do so. (3 marks)

c) Analyse the rest of the design in terms of the design principles we discussed in FIT2099. Write about half a page. (5 marks)

d) Suggest ways to improve the design, and explain why your changes are an improvement. Write a maximum of one page. Include a UML class diagram of your improved design. (7 marks) a)



b)

- Have an abstract message sender
    - In-between controller and SMPTSender
    - Can switch difference message senders in and out

c)

- Point shouldn't know about personal name and family name
    - This should be in a new student class that knows about the students
        - Has location of type point
- Compute distance shouldn't be in the controller
    - Feature Envy
    - Should be in the point class
- Layer of abstraction for SMTPSender
- Attributes should not be public
    - Only the class should know about the attributes, so they should all be private

Question 1. [Total Marks 5]

eXtreme Programming (XP) is an agile development method invented by Kent Beck.

Beck argues that while good design is important, documenting your design, or modelling your design, is not important. He argues that you should focus on “designing always” while coding, rather than producing any design artefacts before implementation.

In about half a page and in your own words, describe some key advantages and disadvantages to this approach, compared to explicitly documenting your design using UML notations such as class and sequence diagrams.

- Documentation provides a map for how you want to code
    - Gives direction
    - If at a hurdle, can refer to the map to help
    - Everyone working knows what to do
    - XP does not allow for this documentation to be made
- Documentation may force a wrong way to implement
    - While coding, it is likely the initial design will change
        - Realise documentation is wrong
        - Better understanding of the problem
        - Refactoring due to adding or removing features
    - Developers may decide to continue down this path, despite it being incorrect
        - Poor design
        - Bugs
        - Unable to extend
    - XP removes this issue
        - Developers don’t feel locked in to the documentation
        - Can always redesign while coding

Question 2. [Total Marks 5]

The Java language uses static typing. That is, the type of every variable and method parameter, and the return types of methods, must be explicitly specified by the programmer.

Other languages, such as Python and JavaScript, use dynamic typing. Variables can hold a value of any type, and methods can return values of any type. At runtime, just before an operation such as an attribute access is performed, the interpreter for the language checks that this will be possible.

Explain in your own words what you think the advantages and disadvantages of static typing are compared to dynamic typing, in the context of developing reliable, maintainable systems. Write about half a page.

- Static typing allows for less errors from the developer
  - They know exactly what is going in and out of every method
    - Easier to read
  - Developers don't get unexpected results or types
  - Earlier detection for errors
  - Refactoring is easier
    - Refactoring incorrectly will throw an error must sooner than dynamic typing
  - Optimised code for the compiler
  - No run-time penalty for determining type
- Static typing is very strict, no flexibility
  - Method must return the correct type
    - i.e. cannot return two different types at any time
  - limits the program

Question 3. [Total Marks 10]

In Java, polymorphism describes the process whereby a subclass overrides a method in a base class (or implements a method defined in an interface). The method code invoked depends on the class of the actual object, not the type specified by the reference.

In the context of the Harry Potter game, explain how the act() method in HPActor and its subclasses uses polymorphism. You may use pseudocode or snippets of Java code, or any kind of diagram if you wish to assist in your explanation. You must clearly explain how polymorphism works in this example.

If you can't remember what the act() method did, instead invent your own example where you would use polymorphism.

Explain why the use of polymorphism in the example is advantageous for writing understandable, maintainable and extensible code.

Take about a page for your answer

- Polymorphism gives the ability to do different things depending on the object
    - In HPActor, some actors act differently than others
        - HouseElf objects move differently to the DeathEaster objects
        - Polymorphism allows this objects with the same base class to have different move methods
- Is easier to understand
    - It is known that every object can act
    - The act ability should be stored in the same place, and called in the same way
    - It doesn't matter what the method does, it should be called and used the same way
- Easier to extend
    - If a new HPActor needs to be added, by simply changing the act method in the new HPActor, it can be called and used in the same way as all the other HPActors when called

Question 4. [Total Marks 1 + 2 + 2 + 1 + 2 + 2 = 10]

Consider the following code for Java classes Watch, Counter, and Driver, similar to those that we saw in lectures at the start of the semester (continued on next page):

```java
public class Watch {

   private Counter milliseconds = new Counter();
   private Counter seconds = new Counter();
   private Counter minutes = new Counter();
   private Counter hours = new Counter();

   public void tick() {
      milliseconds.increment();
      if (milliseconds.getValue() == 1000) {
         milliseconds.reset();
         seconds.increment();
         if (seconds.getValue() == 60) {
            seconds.reset();
            minutes.increment();
            if (minutes.getValue() == 60) {
               minutes.reset();
               hours.increment();
               if (hours.getValue() == 24) {
                  hours.reset();
               }
            }
         }
      }
   }

   public void testWatch(int numTicks) {
      for (int i = 0; i < numTicks; i++) {
         System.out.println(
               String.format("%02d", hours.getValue())
               + ":"
               + String.format("%02d", minutes.getValue())
               + ":"
               + String.format("%02d", seconds.getValue())
               + ":"
               + String.format("%03d", milliseconds.getValue())
         );
         tick();
      }
   }

}
```

```
public class Counter {

   private int value = 0;

   public void reset() {
      value = 0;
   }

   public void decrement() {
      value--;
   }

   public void increment() {
      value++;
   }

   public int getValue() {
      return value;
   }
}

public class Driver {

   public static void main(String[] args) {

      Watch myWatch = new Watch();
      myWatch.testWatch(20000000);
   }

}
```

(a) Give an example of a place in class Watch where there is duplicate code. (1 mark)

(b) Explain how duplicate code can cause problems during software development and maintenance. Give an example of a problem duplicate code could cause, using the code example above. (2 marks)

(c) Suggest a way that the system above could be redesigned so that the duplicated code mentioned in your answer to part (a) would be removed. Explain how this redesign would solve the problem(s) you identified in part (b). (2 marks)

(d) There is a dependency between methods tick() and testWatch(…) in class Watch due to hard-coded constants. Explain what this dependency is. (1 mark)

(e) Explain why excessive dependencies between modules can cause problems during software development and maintenance. Give an example of a problem that such dependencies could cause, using the code example above. (2 marks)

(f) Suggest a way that the system could be redesigned so that the dependency described in your answer to part (d) would be removed. Explain how this redesign would solve the problem(s) you identified in part (e).

a)

- The tick() method has repeated code
    - Object.increment and object.reset is used many times here
- The testWatch() method
    - String has been repeatedly written to print the correct time format

b)

- changing the code in one place means changing code in all others
    - waste time refactoring
    - changing the way a watch resets or increments will cause the tick() method to be changed in multiple places, unnecessarily
- creates implicit dependencies
    - changing one requires changing the others (only if they work the same way)
    - changing how watch resets of increments means being forced to change all instances in tick()
- adding new features is harder
    - trying to add a new counter (e.g. microseconds) means adding in this repeated code again
- makes it harder to read
    - more stuff to read
        - multiple instances of object.increment and object.reset, instead of just one
    - harder to understand
        - is it REALLY the same?
            - Should it be the same
        - Why is it repeated
- Indicates a lack of understanding
    - Did the person writing this really understand what they were doing?

c)

- Tick()
    - Use a for loop to increment all counters
        - Have a maxValue attribute in all counters
        - Compare the current value to maxValue
        - Increment if they are the same
    - Easier to read
        - Should only have one if statement, .reset() and .increment
    - Only need to change the code in one place,
        - there is only one instance of the code
    - No implicit dependencies
        - Do not need to change other locations, if there are none
    - Adding features is easier
        - No need to add code, just add a counter and it will handle it for you
- testWatch()
    - Create a method in Watch
        - This method returns the formatted string
        - Call this before printing
    - Easier to read
        - Can see that the new method is handling all the formatting

    - Only need to change the code in one place
        - No repeated code, so only one location of change is necessary
    - No implicit dependencies
        - Do not need to change other locations, if there are none
    - Adding features is easier
        - All formatting should be handled in the new method, so all hard work is done

d)

- Hard-coded constants are the counters max value, and resets once it reaches them
    - E.g. 24 in hours, 60 in minutes, etc
- TestWatch always prints the current value
- TestWatch uses zero padding to ensure the format is correct
    - e.g. 2 for hours, minutes, and seconds and 3 for milliseconds
- If the constants are changed, this could affect the format for testWatch
    - if hours change from 24 to 1000, then the formatting with the zero padding will be off
    - this formatting is dependant on the log10 of the max value, which has also been hard-coded in testWatch
- Overall, testWatch is assuming that each max value and taking the log of it, so any change to one, may involve changes to the other

e)

- Excessive dependencies mean no freedom
    - Trying to change one aspect may involves making many changes everywhere else
    - Could break the code if any changes are made
    - Introduce new bugs
    - Trying to add something cannot be hard or awkward, since so many things rely too heavily on one another
- E.g. changing a counter's max value
    - If a counter's max value where to be changed in the code, this could affect the formatting
    - the max values are hardcoded, as well as their zero-padding
        - This is the dependency, as the zero-padding is completely based off the max values
    - Changing the max value, but not the zero-padding will introduce new bugs into the code
        - Formatting will be off, will not work as intended

f)

- Have a constant in each counter class that specifies their max value
    - Compare the max value to the current value before resetting and incrementing
    - Have the zero-padding be calculated by using rounding up the log10 of the max value

- This will remove the dependency, as the computer will calculate the required formatting, rather than the developers calculating it themselves, ensuring that it will always be correct, despite any changes made to its max value

Question 5. [Total Marks 5]

It important to manage dependencies in software design. Encapsulation can be used to help limit dependencies.

(a) Explain what is meant by encapsulation in object-oriented design and how it helps to limit dependencies. (2 marks)

Elements of code can be encapsulated at various levels. One example is encapsulation within a method. Variables declared inside a method are local to that method – they cannot be accessed from outside the method.

(b) Give examples of two other levels of encapsulation that are possible in Java. Explain the features of the Java language that allow these levels of encapsulation to be enforced. (3 marks)

a)

- Encapsulation involved isolating a system function or set of data and operations on the data within a module and providing precise specification for the module
    - The concept that access to the names, meanings and values of the methods of a class is entirely separated from access to their realisation
    - The idea that a module has an outside that is distinct from its inside, that it has an external interface and an internal implementation
- It limits dependencies, as everything outside the encapsulation boundaries cannot affect anything inside it
    - Separating the inside from the outside allows for more freedom when creating a new encapsulation boundary
    - Variable names can be the same, if they are in separate encapsulation boundaries
    - Restricts access to its attributes/methods

b)

- Classes
    - Limits access to only the class
    - Only method/attributes of the class can access it, which means nothing else can depend on it
- Packages
    - Limits access to only specific classes
    - Java allows related classes into a subsystem
    - To put a package within a package, the programmer should use dot notation to group them together