Sample Written Answer Exam Questions Part 1

EBU5304 Software Engineering 2018/19

Exam Questions on Design Principles

- Question 3 in the exam will be about Design Principles and Design Patterns
- Design Patterns will be covered in Week 4, they are ways of structuring code that help meet the general idea of Design Principles
- In the 2018/19 exam, part of Question 3 will ask you to write some short code to deal with a programming requirement, and give an explanation of how your code works and why it meets the Design Principles
- The best way to learn how to answer such a question is to practice writing code yourself, compiling and running it
- You have been given some coding exercises to help you do this

Written Answer Exam Questions

- Part of Question 3 in the 2018/19 exam will ask you to write a general explanation of some of the topics covered on Design Principles and Design Patterns
- So you need to write an explanation in English to answer these question parts
- If you want, you can include some example code in your answer to a question like this, but then you must state in English how that code illustrates the general principle
- In exams in previous years, Question 3 was mostly asking for written explanations, but in 2018/19 there will be a question part where you are expected to write some actual code
- The slides here are based on written answer question parts in Question 3 from previous years exams

Learning to Answer Written Answer Exam Questions

- You need to write an explanation that shows you have a good understanding of what the question asks you to explain
- So, before taking the exam, read the notes carefully and make sure you do properly understand them
- An important aspect of getting a good understanding is to do some further reading on the same topics: you have been recommended some text books and websites you can use for this
- Do NOT try to take a short cut by attempting to memorise what was written in the notes without really understanding it
- Practice with writing and running your own code that illustrates the general principles covered will also help you understand them well

How to Answer Written Answer Exam Questions

- You need to write an explanation that shows you have a good understanding of what the question asks you to explain
- Your answer should contain a simple and straightforward explanation of what the question asked you to write about
- You can illustrate the general idea with examples
- You can mention other relevant aspects that are linked to what you are asked to write about
- However, you will not get any marks for writing something that is true but not related to the topic of the question
- If you write something that is incorrect you could lose marks if that suggests you do not really understand something else correct that you wrote

Example Answers

- The example answers to question parts given below are all written in a way that just gives general explanations in English, without illustration with specific examples
- The example answers give relevant aspects that can be mentioned in order to get more marks
- Although these answers have been written in a very thorough way with enough aspects to get full marks, it is possible there are other aspects that could be mentioned that could also get marks.
- If you wanted to answer the question part instead by giving at last some coding example, that would be fine so long as there is at least some explanation in English of how the coding example illustrate the general principle you were asked to write about

Question on re-use

“Instead of writing new code for a module to provide some required service, we may make use of existing code”. Explain in more detail what this statement means, and why it is important. Explain how features of object-oriented programming in Java assist in the *re-use* of existing code for providing new services.

- Making use of existing code saves on programmer time.
- It means the code used will already be well developed and tested for errors.
- The code may be a well-understood standard, making the code that uses it easier to understand (e.g use of standard Java Collections Framework classes).
- Overall size of the code we write is smaller due to lack of repetition
- Note “making use of existing code” means calling existing code, not cut-and-paste copying. So if the code needs to be modified, re-use means all places where it is used share in the benefits of the modification.

“Instead of writing new code for a module to provide some required service, we may make use of existing code”. Explain in more detail what this statement means, and why it is important. Explain how features of object-oriented programming in Java assist in the *re-use* of existing code for providing new services.

- If code was just copied and amended, any modifications would involve finding all places it was copied to, which may be difficult, and modifying each of them, which multiplies the chance of introducing errors with each separate modification.
- The object-oriented principle of programs consisting of classes that define general behaviour of objects as a “template” or “recipe” itself encourages re-use.
- Inheritance gives additional capacity for re-use, as common code will occur in one superclass and is then used in classes which inherit it.
- The dynamic binding principle means code may be written with superclass types as its parameter, and the same code then used with those parameters set to a variety of subclasses.

“Instead of writing new code for a module to provide some required service, we may make use of existing code”. Explain in more detail what this statement means, and why it is important. Explain how features of object-oriented programming in Java assist in the *re-use* of existing code for providing new services.

- Java’s interface type mechanism enables a type to be defined in terms of a set of methods and generalised code to be written which will work for objects of any class where all that is required is use of those methods.
- Generalised code which can be used in a variety of places may use the principle of “parameterisation” meaning an aspect of the code is given by a parameter of an interface type whose value can be varied to provide different behaviours.
- Java also has the concept of “generic typing” meaning generalised code can be written for re-use, instead of particular types of objects to be used, there are “type variables” that represent a range of possible types.

Question on overriding `equals`

Explain what is meant by *overriding equals* when specifying and implementing a Java class. Explain why deciding whether and how to do this is an important decision, and the general contract that needs to be kept to when doing so.

- Every class in Java is a subclass of the class called `Object`.
- That is because a class which is not declared explicitly as extends another class implicitly extends `Object`, so following the chain of extends from any class will always lead to the class `Object`.
- The class `Object` contains several methods, so these methods can be called on objects of any class, one of these methods is `equals` .
- As `Object` is a class, there is code for these methods, but it is possible to override that code by providing a method with the same signature in a class.
- The dynamic binding principle means that when a method is called on an object, code in that object's actual class or its closest superclass which contains code for that method is used to execute the method call.
- The idea of the method `equals` is that the call `obj1.equals(obj2)` returns a `boolean` indicating whether `obj1` and `obj2` refer to objects which are considered equal with the dynamic binding principle meaning this is decided by `obj1`.

Explain what is meant by *overriding equals* when specifying and implementing a Java class. Explain why deciding whether and how to do this is an important decision, and the general contract that needs to be kept to when doing so.

- The default behaviour of `equals` inherited from `Object` is that a call of `obj1.equals(obj2)` returns `true` if and only if `obj1` and `obj2` are aliases, meaning they refer to the same object.
- We may decide it should return `true` if they refer to different objects whose content is otherwise identical, and so write overriding code which provides that.
- This is an important decision because much code, including Java's Collection Framework code, makes use of the method `equals`, with dynamic binding meaning when it does so, it uses any overriding code provided in a class.
- Care must be taken to ensure that when `equals` is overridden, it still performs in a way compatible with the expectations of an equality relationship, the "general contract" for `equals`.
- The required property of symmetry means `obj1.equals(obj2)` should give the same as `obj2.equals(obj1)`
- The property of transitivity means if `obj1.equals(obj2)` is `true` and `obj2.equals(obj3)` is `true`, then also `obj1.equals(obj3)` should be `true`.

Question on Inheritance and the Liskov Substitution Principle

Another issue in Java and similar programming languages to do with variables referring to objects is that a variable has a declared type, but it can be set to refer to an object whose actual type is a subtype of that declared type. Explain the advantages which this gives, but also the problem which led to the *Liskov Substitution Principle (LSP)* being declared as an important guideline for good practice.

- This issue comes from the idea of inheritance which enables us to write classes which are declared as variants of other classes (subclasses) and are expressed only in terms of how they differ from the class they are derived from (the superclass).
- The difference may be additional methods to those of the superclass, but it can also be different code for an existing method in the superclass (overriding).
- This has the advantage of reducing duplication of code, keeping code which performs common operations in the superclass and not repeated in its subclasses.
- The “actual type” of an object is the class of its constructor, but a variable declared of a superclass type can be set to refer to an object whose actual type is a subclass of that type.

Another issue in Java and similar programming languages to do with variables referring to objects is that a variable has a declared type, but it can be set to refer to an object whose actual type is a subtype of that declared type. Explain the advantages which this gives, but also the problem which led to the *Liskov Substitution Principle (LSP)* being declared as an important guideline for good practice.

- This gives another way of avoiding duplication of code, as methods whose parameters are declared of a superclass type may be used with those parameters set to refer to objects whose actual type is a subclass of that type.
- So this gives us generalised code, which can be re-used instead of having to write new code to handle object of the subclass types.
- The generalised code will be for operations that only require the methods of the superclass, but those methods could be overridden in the actual class of its arguments. The code that is used to execute them is the overriding code in the subclass.
- This is known as “dynamic binding”, as the decision on which code to use has to be made at run-time rather than compile time.

Another issue in Java and similar programming languages to do with variables referring to objects is that a variable has a declared type, but it can be set to refer to an object whose actual type is a subtype of that declared type. Explain the advantages which this gives, but also the problem which led to the *Liskov Substitution Principle (LSP)* being declared as an important guideline for good practice.

- However, to override a method in terms of code that will compile, all that is needed is to give a new method with the same signature as the overridden method.
- The Liskov Substitution Principle states that overriding should be done only with methods whose operations are compatible with the specification of the methods being overridden.
- This is so that any reasoning in the design and implementation of generalised code will still apply when that code is used with arguments of a subclass.
- If the LSP were not observed and it was not known or planned that when code was running its arguments refer to objects of a subclass, the code could perform in an unanticipated, even dangerous or insecure, way.

Question on aliasing

Explain what is meant by **aliasing** in Java programming, and why care needs to be taken over the possibility of aliasing occurring. Explain how aliasing can lead to the problem of **“exposing the representation”** and why exposing the representation could have serious consequences

- In Java, variables of an object type “refer to” objects .
- That means they store the address in memory of the data of the object rather than the actual data.
- As a consequence, if one variable is assigned the value of another, it results in the two variables referring to the same object, that is what “aliasing” means.
- If two variables are aliases it means if there is a change of the state of the object referred to by a method call on one of the variables, the object referred to by the other variable will also be changed, as they are the same object.

Explain what is meant by **aliasing** in Java programming, and why care needs to be taken over the possibility of aliasing occurring. Explain how aliasing can lead to the problem of **“exposing the representation”** and why exposing the representation could have serious consequences

- If you were not aware that one variable is aliased by another, it could lead to what would be a hard to understand bug, as an object would seem to change for no reason when it was not referred to in the code.
- “Exposing the representation” means a situation where a variable that is a field inside an object is aliased by a variable that is outside the object.
- This could occur if a direct reference to another object is passed in through a constructor or a method call and a field variable is set to it
- or if the value of a field variable of an object types is passed as the return value of a method call made on the object.

Explain what is meant by **aliasing** in Java programming, and why care needs to be taken over the possibility of aliasing occurring. Explain how aliasing can lead to the problem of **"exposing the representation"** and why exposing the representation could have serious consequences

- An important principle of object-oriented programming is that the internal state of an object should be changed only by the code in its own class.
- If the representation is exposed, the state could be changed by other code referring to an internal object that is part of the state through the alias.
- This would make it much harder to be certain that the code is working correctly, and give incorrect behaviour if it was not realised this was happening.
- It could also lead to the danger of breaches of security, as it may result in other code gaining access to data that is not meant to be directly available.