Aston University

School of Engineering and Applied Science

CS2020 - Software Engineering

Stage Two Examination

CLOSED BOOK

Date: 22nd January, 2016
Time: 14:00 – 17:00
Duration: 3 hours

Instructions to Candidates

1. **Answer ALL questions from Section A. (40 marks)**
2. **Section A contains TEN questions.**
3. **Answer THREE questions from Section B. (60 marks)**
4. **Section B contains FOUR questions, each question is worth 20 marks.**
5. **Use of calculators IS NOT allowed.**

Materials provided

1. **Answer booklets**
2. **Appendix --- API specification for a Java class.**

This exam paper cannot be removed from the exam room

Section A — Answer ALL questions

1. EXPLAIN the conceptual difference between a **model** and a **diagram** as used in software development, referring to UML models and UML diagrams as examples.

(3 marks)

2. a) **INVEST** is an acronym made up of six qualities of a good **user story**. STATE any FOUR of these six qualities.

(2 marks)

b) Consider the following **user story** in the development of an alarm clock:

> As a user, I would like to be able to set the alarm for a specific day only so that I do not get disturbed the following day.

Choose TWO of the four INVEST qualities you named in part (a) and BRIEFLY EVALUATE this **user story** for these two qualities.

(2 marks)

3. BRIEFLY EXPLAIN why it is not necessary to include **state** names in a **state machine diagram**. Also, GIVE A REASON why it could be useful to include **state** names.

(3 marks)

4. a) DEFINE the terms **entity object**, **boundary object** and **control object** as used in object-oriented systems analysis.

(3 marks)

b) For EACH of these three types of object, IDENTIFY an object of this type in the context of an application to manage maps of industrial sites.

(3 marks)

5. a) DESCRIBE the key difference between a **centralised** and **distributed** version control system (VCS).

(2 marks)

b) NAME ONE popular distributed VCS and ONE popular centralised VCS system.

(2 marks)

6. a) **SOA** is a system architecture. What does **SOA** stand for? (1 mark)

   b) NAME a kind of software system for which **SOA** would be suitable.

   (1 mark)

   c) STATE ONE reason why **SOA** is needed for the kind of software system you named in part (b).

   (1 mark)

7. Consider the following scenario:

   An application A includes source code C_1 and C_1 has been tested thoroughly.
   The application A has just been extended with new source code C_2.
   C_1 is independent of C_2.

   a) Is there a need to perform **regression testing** on C_1? (1 mark)

   b) Making reference to what is meant by **regression testing**, briefly EXPLAIN your answer to part (a).

   (3 marks)

8. a) EXPLAIN what is meant by a **framework** in software engineering.

   (2 marks)

   b) NAME a **framework** in the Java language that is designed for **unit testing**.

   (1 mark)

   c) STATE the main purpose of an **Object Relational Mapping (ORM)** framework such as **Hibernate**.

   (1 mark)

   d) STATE what is meant by **lazy materialization** within the context of a **persistence framework**.

   (1 mark)

9. a) Briefly EXPLAIN what is meant by EACH of the following qualities of software design:

   - **Maintainable**
   - **Reliable**

   (2 marks)

   b) A residential property management company manages various types of residential properties (eg houses, flats) on behalf of their clients. When there is a fault within a property (eg a faulty light switch), the resident reports the fault to a property manager by phone, email or post. Upon receipt of a fault report, the property manager arranges repairs to be carried out as soon as possible.

   Suppose you have been tasked to design an online computer system for residents to report faults to property managers. NAME ONE quality of design for this system that is very important and cannot be compromised. Briefly EXPLAIN your answer.

   (2 marks)

10. Consider the following class `Person` which models some basic behaviours of a human being. A female person can give birth to a child, and the mother of the child will be the person who gave birth to that child.

```
1  /** Class Person models some basic behaviours of a human being. */
2  public class Person {
3     private Person mother;     // The mother of this Person object
4     private String name;       // The name of this Person
5     private boolean isMale;    // true for male
6
7     /** Constructor: creates a new Person object */
8     public Person(boolean isMale, String name,  Person mother) {
9          this.isMale = isMale;
10         this.name = name;
11         this.mother = mother;
12    }
13
14    /** This person gives birth to a child.
15      *  The child is returned upon creation. */
16    public Person givesBirth(String childName, boolean isMaleChild) {
17         // The mother of the child is 'this' object.
18         return new Person(isMaleChild, childName, this);
19    }
20
21    // Other methods omitted
22 }
```

(question continues on next page. . .)

(Question 10 continued. . .)

a) STATE what is meant by **precondition** in the context of a method.

(1 mark)

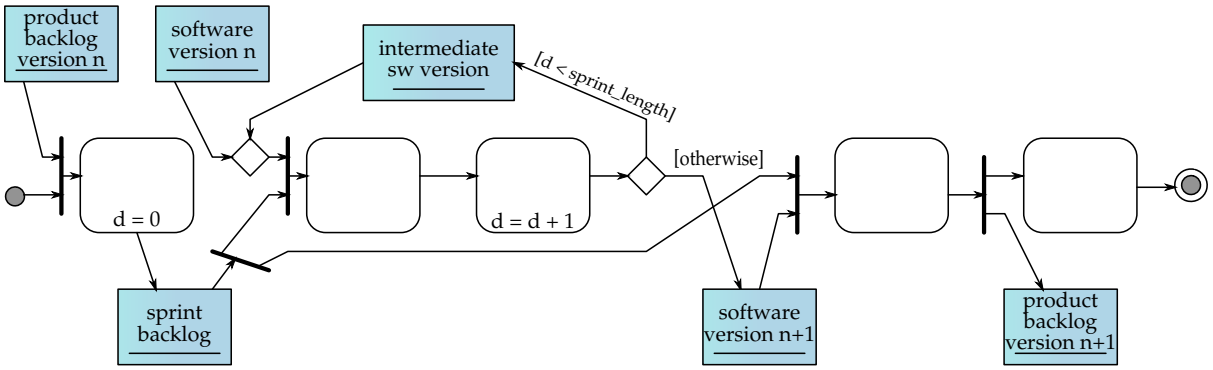
b) WRITE an `assert` statement in Java to introduce a suitable **precondition** check to method `givesBirth` in class `Person`. Making use of the given line numbers, state where your `assert` statement should be inserted.

(3 marks)

END OF SECTION A

Section B — Answer any THREE questions of the next four

11.a) The following incomplete activity diagram illustrates ONE **sprint** of the **Scrum** agile software development process.



SUPPLY the missing descriptions of the FIVE activities. (5 marks)

b) STATE the recommended length of a Scrum sprint. (1 mark)

c) Briefly EXPLAIN the roles of the Scrum **product owner** and the Scrum **master**.

(2 marks)

d) Recall the Agile manifesto:

individuals and interactions	over	processes and tools
working deliverables	over	comprehensive documentation
customer collaboration	over	contract negotiation
responding to change	over	following a plan

For EACH of the four points of the manifesto:

i) DESCRIBE ONE type of activity that a Scrum team would undertake and which illustrates this point, and

(4 marks)

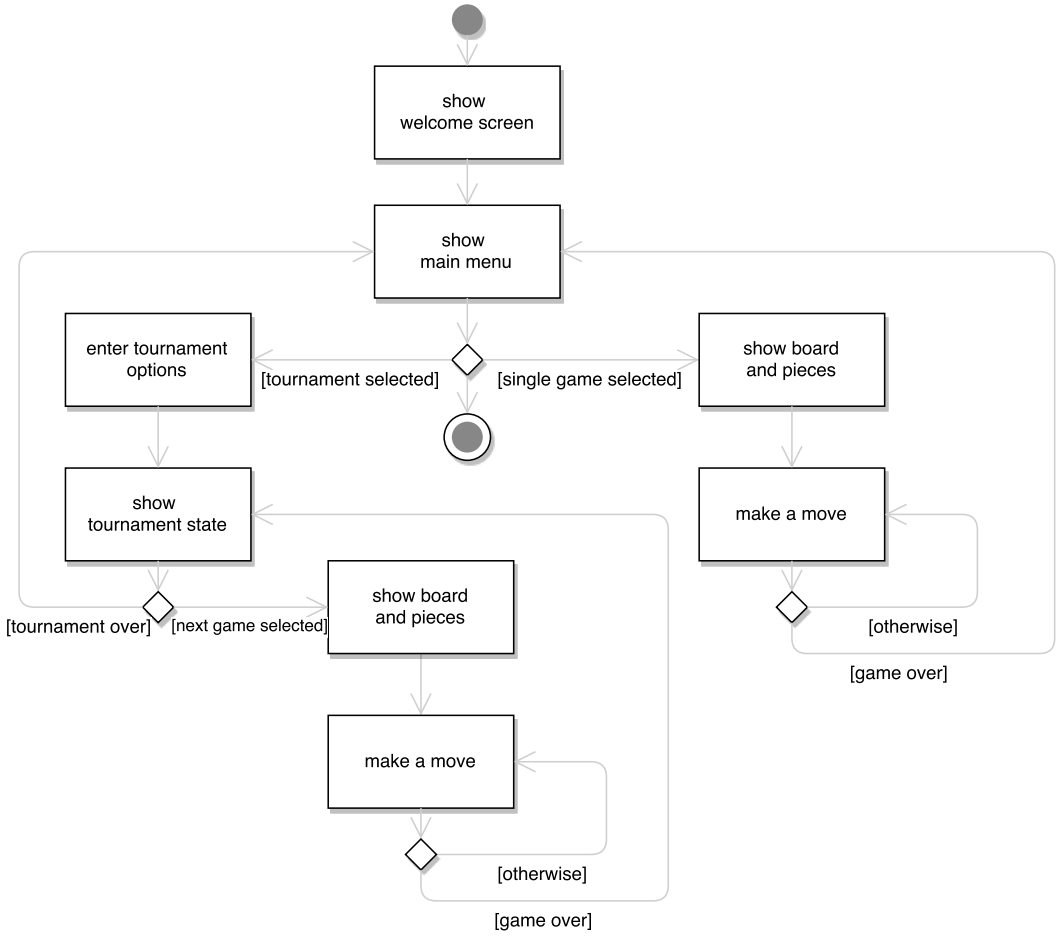
ii) JUSTIFY this activity. (4 marks)

Choose a different activity to illustrate each point, ie FOUR *activities overall.*

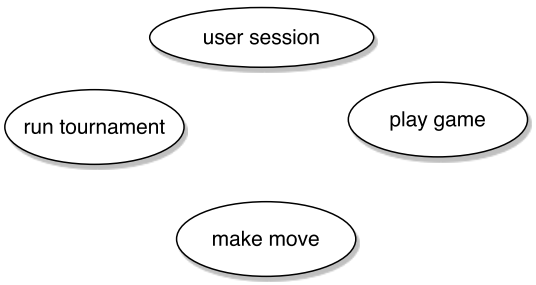
e) DESCRIBE TWO important commonalities and TWO important differences between agile development processes such as Scrum and the Unified Process.

(4 marks)

12. Consider the behaviour of a game application described by the following **activity diagram**:



The following is an incomplete **use case diagram** of this game application:



a) DRAW the above **use case diagram** and ADD actors into it, appropriately connecting them with use cases.

(2 marks)

b) DRAW any appropriate relationships between use cases in your copy of the above use case diagram.

(5 marks)

(question continues on next page. . .)

(Question 12 continued. . .)

c) Consider the following partially filled use case description form:

Use case number: 2	**Name**: Run Tournament
Goal: Set up and carry out a game tournament.	
Brief description: Involves setting up a tournament and running a set of games and reporting on progress and results in between.	
Frequency of execution:	
Scalability:	
Preconditions:	
Postconditions:	
Primary path:	
Alternatives:	
Exceptions:	

In the following tasks, you will continue completing this form.

In your answers, assume that a tournament can run only with 4 or more players and that players may abandon a tournament or new players can join a tournament that is underway. You can make any further reasonable assumptions on the details of the game and users' preferences.

i) Some of the fields in the form refer to **non-functional** requirements. IDENTIFY these fields. (1 mark)

ii) STATE the **non-functional** requirements that should be added to the fields identified in part (i). (2 marks)

iii) STATE ONE **precondition** and ONE **postcondition** that should be added to the above form. (4 marks)

iv) DESCRIBE the **primary path** of this use case.

Use a format suitable for the above form. (2 marks)

v) DESCRIBE ONE **alternative** and ONE **exception** of this use case.

Use a format suitable for the above form. (4 marks)

13. a) Consider the following Java class `MyRandom`.

Listing 1: Class `MyRandom`

```
1  import java.util.Random;
2  /** Models a basic random number generator */
3  public class MyRandom {
4      private Random generator;
5      private static MyRandom myRandom = new MyRandom();
6
7      private MyRandom() {
8          generator = new Random();
9      }
10
11     public static MyRandom getInstance() {
12         return myRandom;
13     }
14
15     public void setSeed(int seed) {
16         generator.setSeed(seed);
17     }
18
19     public int nextInt() {
20         return generator.nextInt();
21     }
22 }
```

i) NAME the object-oriented **design pattern** used in class `MyRandom`.

(1 mark)

ii) Making use of a detailed UML class diagram, OUTLINE the general form of the object-oriented design pattern used in class `MyRandom`.

(3 marks)

iii) Briefly DESCRIBE the purpose of the object-oriented design pattern used in class `MyRandom`.

(2 marks)

b) Making reference to what is meant by **cohesion**, EXPLAIN what is meant by **specialisation cohesion**. What does **specialisation cohesion** measure?

(3 marks)

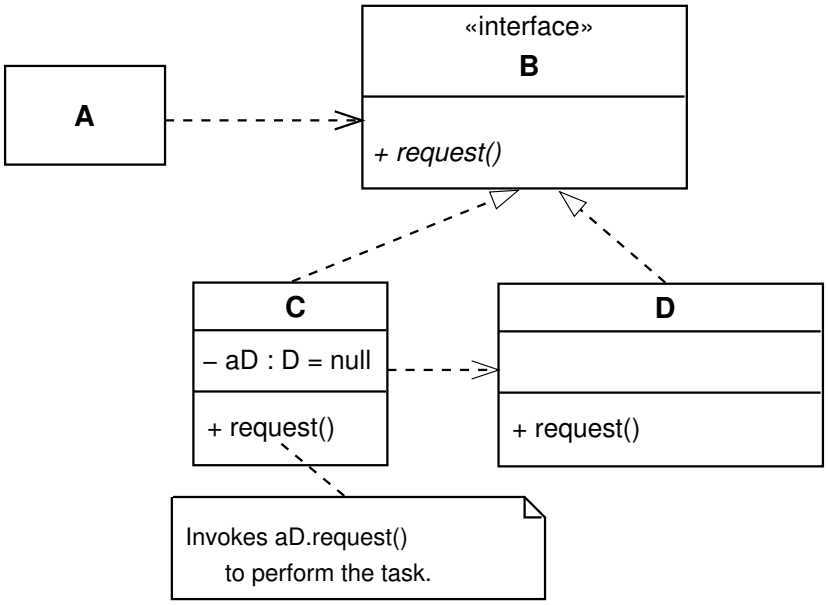
c) Consider the API specification for the Java class `java.util.Stack` in the Appendix. EVALUATE the design of `java.util.Stack` in terms of **specialisation cohesion**.

(4 marks)

(question continues on next page. . .)

(Question 13 continued. . .)

d) Consider the following detailed UML **class diagram** which shows the general form of an object-oriented design pattern as defined by the "Gang of Four".



i) NAME the design pattern described by the given detailed UML **class diagram**.

(1 mark)

ii) GIVE a suitable name to EACH of the classes A, B, C and D shown in the given detailed UML **class diagram**.

(4 marks)

iii) Making reference to its intended purpose, briefly DESCRIBE ONE application of the given "Gang of Four" design pattern.

(2 marks)

14.a) Consider a Java application which contains the following outline Java code with six classes and one interface. This Java application has been developed using an **agile** software development methodology. The following Java classes and interface were the outcome of a **sprint**.

Listing 2: Class `Borrower`

```
1  import java.util.Date;
2
3  public class Borrower implements Returner {
4      private static int BORROW_LIMIT = 10;
5
6      private Date dateOfBirth;
7      private BorrowableItem[] borrowed;
8      private int outstandingFine;
9      private String contactDetails;
10
11     // other field definitions omitted
12
13     public Borrower(String name, String contactDetails) {
14         borrowed = new BorrowableItem[BORROW_LIMIT];
15         // other details omitted
16     }
17
18     public int returnItem(BorrowableItem item) {
19         // details omitted
20     }
21
22     public void payFine(int amount) {
23         // details omitted
24     }
25
26     private boolean isOver18() {
27         // details omitted
28     }
29
30     // Other method details omitted
31 }
```

Listing 3: Class `Library`

```
1  public class Library {
2      private BorrowableItem[] catalogue;
3
4      // other details omitted
5  }
```

(question continues on next page. . .)

(Question 14 continued. . .)

Listing 4: Class `BorrowableItem`

```
1  public abstract class BorrowableItem {
2      private Borrower borrower;
3      private String title;
4
5      // other field definitions omitted
6
7      public boolean isReturned() {
8          // details omitted
9      }
10
11     // other method details omitted
12 }
```

Listing 5: Class `DVD`

```
1  public class DVD extends BorrowableItem {
2    // other details omitted
3  }
```

Listing 6: Class `Book`

```
1  public class Book extends BorrowableItem {
2    // other details omitted
3  }
```

Listing 7: Class `Staff`

```
1  public class Staff implements Returner {
2    // other details omitted
3  }
```

Listing 8: Interface `Returner`

```
1  public interface Returner {
2      public abstract int returnItem(BorrowableItem item);
3  }
```

DRAW a detailed UML **class diagram** containing ALL of the above classes and interface, showing ALL attributes, operations and class relationships with appropriate multiplicity defined in the given Java code.

*Your solution should also include the **import** relation stated in class* `Borrower`*.*

(10 marks)

(question continues on next page. . .)

(Question 14 continued. . .)

b) Consider a later **sprint** which focuses on improving the communication between a borrower and the library. One of the user stories identified is to enable automatic sending of reminders to a borrower about the due date of his/her borrowed item(s).

i) Briefly DESCRIBE what is meant by **refactoring**. (2 marks)

ii) NAME a type of **refactoring** that is needed for the above partial definition of class `Borrower` in Listing 2 so as to prepare it for use as the code base in the later **sprint**. (1 mark)

iii) Making use of the given line numbers, DESCRIBE the modifications required to the given code base when applying the type of **refactoring** required in part (ii).

(2 marks)

iv) Making reference to HOW and WHY a code base evolved during a typical **agile** software development, briefly EXPLAIN why it is beneficial to perform the type of **refactoring** required in part (ii). (5 marks)

END OF EXAMINATION PAPER

Appendix — API Specification for Class `java.util.Stack`

compact1, compact2, compact3

java.util

Class Stack<E>

java.lang.Object
  java.util.AbstractCollection<E>
    java.util.AbstractList<E>
      java.util.Vector<E>
        java.util.Stack<E>

All Implemented Interfaces:

Serializable, Cloneable, Iterable<E>, Collection<E>, List<E>, RandomAccess

---

```
public class Stack<E>
extends Vector<E>
```

The `Stack` class represents a last-in-first-out (LIFO) stack of objects. It extends class `Vector` with five operations that allow a vector to be treated as a stack. The usual `push` and `pop` operations are provided, as well as a method to `peek` at the top item on the stack, a method to test for whether the stack is `empty`, and a method to `search` the stack for an item and discover how far it is from the top.

When a stack is first created, it contains no items.

A more complete and consistent set of LIFO stack operations is provided by the `Deque` interface and its implementations, which should be used in preference to this class. For example:

```
   Deque<Integer> stack = new ArrayDeque<Integer>();
```

Since:

JDK1.0

See Also:

Serialized Form

Field Summary

Fields inherited from class java.util.Vector

capacityIncrement, elementCount, elementData

Fields inherited from class java.util.AbstractList

modCount

Constructor Summary

Constructors

Constructor and Description
Stack() Creates an empty Stack.

Method Summary

All Methods **Instance Methods** **Concrete Methods**

Modifier and Type	Method and Description
boolean	**empty**() Tests if this stack is empty.
E	**peek**() Looks at the object at the top of this stack without removing it from the stack.
E	**pop**() Removes the object at the top of this stack and returns that object as the value of this function.
E	**push**(E item) Pushes an item onto the top of this stack.
int	**search**(Object o) Returns the 1-based position where an object is on this stack.

Methods inherited from class java.util.Vector

add, add, addAll, addAll, addElement, capacity, clear, clone, contains, containsAll, copyInto, elementAt, elements, ensureCapacity, equals, firstElement, forEach, get, hashCode, indexOf, indexOf, insertElementAt, isEmpty, iterator, lastElement, lastIndexOf, lastIndexOf, listIterator, listIterator, remove, remove, removeAll, removeAllElements, removeElement, removeElementAt, removeIf, removeRange, replaceAll, retainAll, set, setElementAt, setSize, size, sort, spliterator, subList, toArray, toArray, toString, trimToSize

Methods inherited from class java.lang.Object

finalize, getClass, notify, notifyAll, wait, wait, wait

Methods inherited from interface java.util.Collection

parallelStream, stream

Constructor Detail

Stack

```
public Stack()
```

Creates an empty Stack.

Method Detail

push

```
public E push(E item)
```

Pushes an item onto the top of this stack. This has exactly the same effect as:

```
addElement(item)
```

Parameters:

`item - the item to be pushed onto this stack.`

Returns:

`the item argument.`

See Also:

`Vector.addElement(E)`

pop

```
public E pop()
```

Removes the object at the top of this stack and returns that object as the value of this function.

Returns:

`The object at the top of this stack (the last item of the Vector object).`

Throws:

`EmptyStackException - if this stack is empty.`

peek

```
public E peek()
```

Looks at the object at the top of this stack without removing it from the stack.

Returns:

`the object at the top of this stack (the last item of the Vector object).`

Throws:

`EmptyStackException - if this stack is empty.`

empty

```
public boolean empty()
```

Tests if this stack is empty.

Returns:

true if and only if this stack contains no items; false otherwise.

search

```
public int search(Object o)
```

Returns the 1-based position where an object is on this stack. If the object `o` occurs as an item in this stack, this method returns the distance from the top of the stack of the occurrence nearest the top of the stack; the topmost item on the stack is considered to be at distance 1. The `equals` method is used to compare `o` to the items in this stack.

Parameters:

`o - the desired object.`

Returns:

`the 1-based position from the top of the stack where the object is located; the return value -1 indicates that the object is not on the stack.`