EECS3311 Deferred Exam Fall 2022

Student ID:	**Signature/Student Name:**
Section:	**Instructor:**

©2022 York University
Department of Electrical Engineering and Computer Science
EECS3311 – Software Design
Instructor: Ilir Dema

Instructions:

- Please double check the instructor and section information, there are three different sections.
- You have **180 minutes** to complete the exam.
- If you separate the pages, make sure your names and student IDs are on the top of every page.
- Unauthorized duplication or archival of these questions is not permitted.
- If information appears to be missing from a question, make a reasonable assumption, state your assumption, and proceed. Do not simplify the question.
- You must answer questions in a clear piece of paper and send your solution to your instructor.
- **Illegible answers receive NO point.**

Question	Mark	Points
Q1 (MCQ)		10
Q2		10
Q3		15
Q4		10
Q5		15
Q6		10
Q7		20
Q8		10
Total:		

Question 1: Multiple Choice Questions (10 points)

1) Which of the following design patterns defines a one-to-many relationship between an object and other objects that are notified about changes in its state: Circle one. [1 point]

a. Proxy pattern.

b. Decoractor pattern.

c. Visitor pattern.

d. Observer pattern.

e. Facade Pattern.

f. Composite Pattern.

g. None of the above.

2) Double Dispatch is used by which design pattern. [1 point]

a. Proxy pattern.

b. Decoractor pattern.

c. Visitor pattern.

d. Observer pattern.

e. Facade Pattern.

f. Composite Pattern.

g. None of the above.

3) Assess the correctness of the following statements. Write T or F in the space next to each sentence. [3 points]

Statement	True / False
The observer pattern is infrequently used in UI-driven systems.	
In some circumstances design patterns can decrease system understandability.	
The command pattern is often used when undo/redo functionality is desired.	
Coupling is a negative software property while cohesion is a positive quality.	
The open/close principle means software should be open for modification, but closed for extension	
State pattern cannot be used together with visitor pattern	

4) Imagine that you are implementing a file system. The main abstractions in your design would be files and directories. Directories can contain zero or more files or directories. You want to treat directories and files in a uniform way, e.g., both will have a name and a size, and will provide operations to stream content in and out, and to list children. What design pattern could be used to achieve this design? [1 point]

a. Proxy pattern.

b. Decoractor pattern.

c. Visitor pattern.

d. Observer pattern.

e. Facade Pattern.

f. Composite Pattern.

5) Imagine that you are asked to make your code work with a broad set of objects that belong to a sophisticated library or framework. What design pattern is being used? [2 points]

a. Proxy pattern.

b. Decoractor pattern.

c. Visitor pattern.

d. Observer pattern.

e. Facade Pattern.

f. Composite Pattern.

6) Consider the following original method:

```
static int findMax(int a[], int aLength) {
  // if a is null , throw NullPointerException
  // if aLength <= 0, undefined behaviour
  int max = a[0];
  for (int i=0; i < aLength; i++) {
    if (a[i] > max) {
      max = a[i];
    }}
  return max;}
```

Here is a mutant of the above method:

```
static int findMax(int a[], int aLength) {
  // if a is null , throw NullPointerException
  // if aLength <= 0, undefined behaviour
  int max =a[0];
  for (int i=0; i < aLength; i++) {
    if (a[i] > max) {
      max = a[1];// <-- mutation
      }}
  return max;}
```

Which of the following test cases can weakly kill (but does not strongly kill) the mutant? [1 point]

a. a={}, aLength =0.

b. a={10,1,2}, aLength =3.

c. a={1,10,2}, aLength =3.

d.a={1,2,10}, aLength =3.

Which of the following test cases can strongly kill the mutant? [1 point]

a. a={}, aLength =0.

b. a={10,1,2}, aLength =3.

c. a={1,10,2}, aLength =3.

d.a={1,2,10}, aLength =3.

Question 2: Software Architecture (10 points)

Consider the following project description:

The purpose of the project is to develop a reusable system architecture for oscilloscopes. An oscilloscope is an instrumentation system that samples electrical signals and displays pictures (called traces) of them on a screen. The architecture should address the following functions: signal transformers serve to condition external signals, acquisition transformers derive digitized waveforms from these signals, display transformers convert these waveforms into visual data. Additionally, the oscilloscopes should perform measurements on the signals, and also display these on the screen.

- i) Which architectural style would you recommend for this system [2 points]?

- ii) Sketch the resulting architectural view (show components, connectors, and provide a brief description of your model) [6 points].

- iii) What are the main advantages and disadvantages of this style [2 points]?

Question 3: Software Architecture II (15 points)

For each of the three systems below, answer the following questions [5 points per system for a total of 15 points]:

- a) Which architectural style is most appropriate for the given system [2 points]?
- b) Give a sketch of a sample architecture for the given system. Explain the interaction between the parts of the architecture [2 points].
- c) Discuss the reason for selecting a given style for the given system. [1 point].

Consider the following three systems:

1. A development environment capable of integrating a set of tools (e.g., compiler, editor, debugger, etc.) produced by different vendors.

2. An application with a graphical user interface providing several editable views on the application data; new views should be easy to add at any time and the views should be automatically kept in synch with the application data.

3. An operating system with basic services such as interrupt and exception handling, thread scheduling and dispatching; resource management services such as object management, virtual memory management, I/O management, and security monitor; and, finally, utility programs such as command shell and administration tools.

Question 4: Design Pattern (10 points)

Imagine that you would like to implement utility programs such as **ls** and **chmod** that need to iterate over file/directory structures from the previous question and perform operations on the visited nodes. In your design, you would like to avoid the need to extend the interface of the classes representing the file/directory structure whenever you add a new utility program.

- i) What design pattern could be used to achieve this design? [3 points]
- ii) Explain your design by giving a class diagram and explaining the classes and key methods and interactions. Describe the new classes in detail. [5 points]
- iii) Using pseudo-code, describe how to implement a utility that lists all files and directories in a particular directory. [2 points]

Question 5: Design By Contract (15 points)

Suppose we implement the `InsertionSort` algorithm with the following code. `InsertionSort` is a simple sorting algorithm that works similar to the way you sort playing cards in your hands. The array is virtually split into a sorted and an unsorted part. Values from the unsorted part are picked and placed at the correct position in the sorted part. Code are from `https://www.geeksforgeeks.org/insertion-sort/`.

```java
public class InsertionSort {
/*Function to sort array using insertion sort*/
  void sort(int arr[]){
  int n = arr.length;
    for (int i = 1; i < n; ++i) {
      int key = arr[i];
      int j = i - 1;

/* Move elements of arr[0..i-1], that are greater than key, to one position ahead of their current position */
      while (j >= 0 && arr[j] > key) {
        arr[j + 1] = arr[j];
        j = j - 1;
      }
      arr[j + 1] = key;
    }
  }

/* A utility function to print array of size n*/
static void printArray(int arr[]){
  int n = arr.length;
  for (int i = 0; i < n; ++i)
    System.out.print(arr[i] + " ");
    System.out.println();
}

// Driver method
public static void main(String args[]){
  int arr[] = { 12, 11, 13, 5, 6 };
  InsertionSort ob = new InsertionSort();
  ob.sort(arr);
  printArray(arr);
  }
};
```

Please finish the following contracts for the methods in class `InsertionSort`:

(a) Pre-condition for sort(): “arr[]” should not be null or empty. [2 points]

(b) Pre-condition for sort(): “arr[]” should not be sorted. [2 point]

(c) Post-condition for sort(): after executing `sort()`, the elements in “`arr`” are sorted in ascending order. [2 points]

(d) Post-condition for sort(): the elements in “`arr`” are the same before and after executing `sort())`. [3 points]

(e) Loop Invariant for while statement (line 10): during the process of sorting, the already sorted ones should be sorted in ascending order. [3 points]

(f) Loop Invariant of while statement (line 10): during the process of sorting, elements in “`arr`” are not changed. [3 points]

Question 6: Design Pattern (10 points)

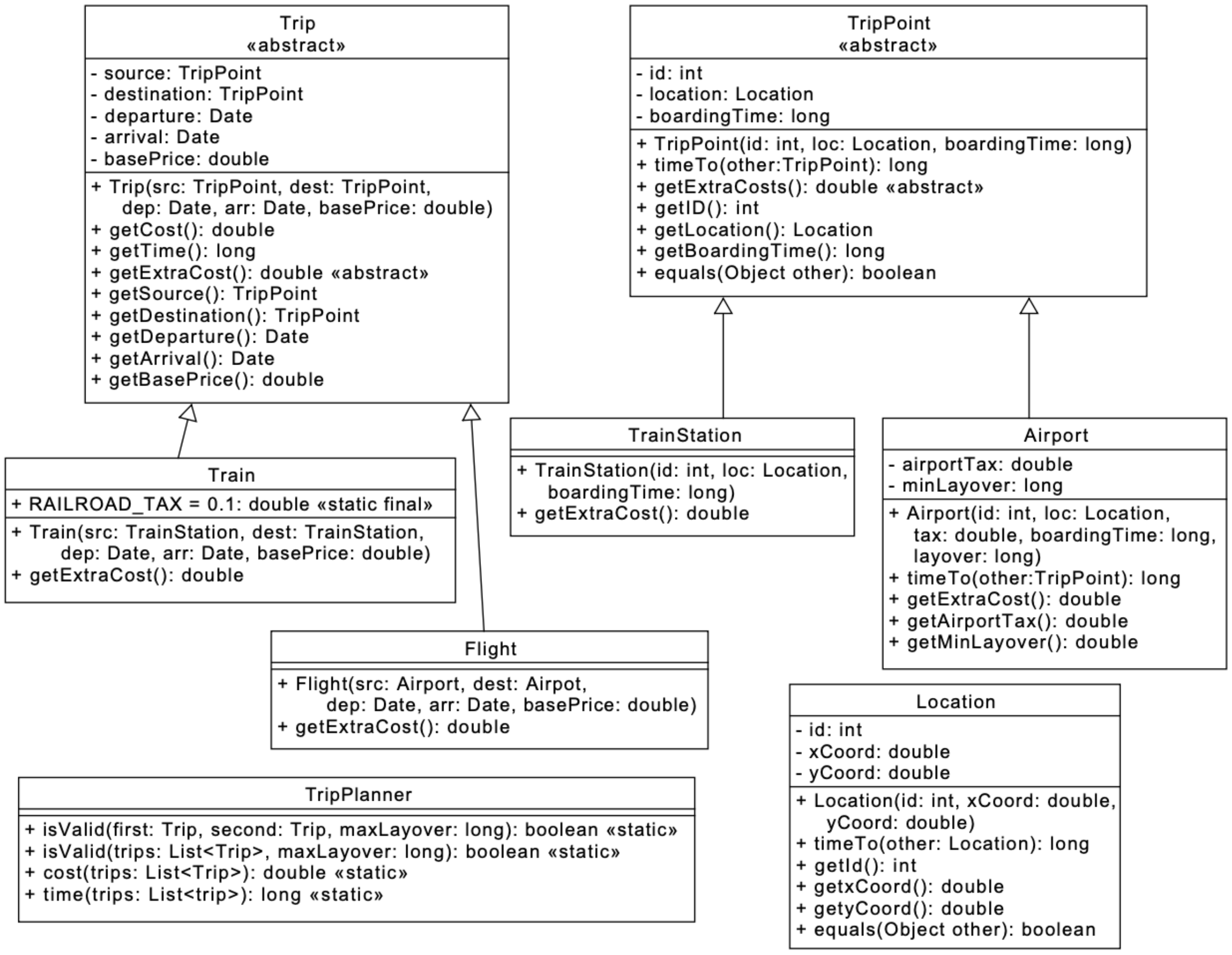
Given the following code snippet:

```
OutputStream os =
new BufferedOutputStream (
new FileOutputStream (
new File (" EECS3311final . tex " )));
```

- i) What design pattern is being used [4 points]?
- ii) Why would a designer use this pattern [2 points]?
- iii) What benefit does this pattern provide to clients of these APIs [2 points]?
- iv) What is a downside of this pattern [2 points]?

Question 7: Class Diagram (20 points)

Consider the following UML diagram:


Trip
«abstract»
- source: TripPoint
- destination: TripPoint
- departure: Date
- arrival: Date
- basePrice: double
+ Trip(src: TripPoint, dest: TripPoint, dep: Date, arr: Date, basePrice: double)
+ getCost(): double
+ getTime(): long
+ getExtraCost(): double «abstract»
+ getSource(): TripPoint
+ getDestination(): TripPoint
+ getDeparture(): Date
+ getArrival(): Date
+ getBasePrice(): double
TripPoint
«abstract»
- id: int
- location: Location
- boardingTime: long
+ TripPoint(id: int, loc: Location, boardingTime: long)
+ timeTo(other:TripPoint): long
+ getExtraCosts(): double «abstract»
+ getID(): int
+ getLocation(): Location
+ getBoardingTime(): long
+ equals(Object other): boolean
Train
+ RAILROAD_TAX = 0.1: double «static final»
+ Train(src: TrainStation, dest: TrainStation, dep: Date, arr: Date, basePrice: double)
+ getExtraCost(): double
TrainStation
+ TrainStation(id: int, loc: Location, boardingTime: long)
+ getExtraCost(): double
Airport
- airportTax: double
- minLayover: long
+ Airport(id: int, loc: Location, tax: double, boardingTime: long, layover: long)
+ timeTo(other:TripPoint): long
+ getExtraCost(): double
+ getAirportTax(): double
+ getMinLayover(): double
Flight
+ Flight(src: Airport, dest: Airpot, dep: Date, arr: Date, basePrice: double)
+ getExtraCost(): double
Location
- id: int
- xCoord: double
- yCoord: double
+ Location(id: int, xCoord: double, yCoord: double)
+ timeTo(other: Location): long
+ getId(): int
+ getxCoord(): double
+ getyCoord(): double
+ equals(Object other): boolean
TripPlanner
+ isValid(first: Trip, second: Trip, maxLayover: long): boolean «static»
+ isValid(trips: List<Trip>, maxLayover: long): boolean «static»
+ cost(trips: List<Trip>): double «static»
+ time(trips: List<trip>): long «static»

Study this diagram carefully, until you are certain you understand the design, before you move on to the implementation. In this system, a `Trip` describes exactly one `Flight/Train ride/etc`. Each Flight connects two different `Airports`. Each Train ride connects two different `TrainStations`. In addition to the `basePrice` of each `Trip`:

- each Airport on the way, charges an airportTax (different for each Airport), recorded in the same units as the basePrice;
- each Train ride on the way, charges a RAILROAD TAX (the same for any train ride), recorded as a percentage of a basePrice.

As for travel time, in addition to the duration of each trip (the difference between departure and arrival times), you need to consider that:

- each TripPoint — an Airport or a TrainStation — has a boarding time associated with it, which is the minimum time required from arriving there to boarding a plane/train/etc.
- if there are two Flights that connect in the same Airport, then there is a minimum layover time (specified for each airport) required to make a connecting flight;
- transfer time between two different TripPoints is the time required to get from one corresponding Location to the other, which can be calculated via a complicated algorithm that involves Google Maps, and which is implemented in Location.timeTo.

Your task in this question is to complete the implementation of the classes below, so that they:

1 correspond to the UML diagram,

2 follow the specifications above, and

3 implement the methods according to their Javadocs (where provided), i.e., the class declarations, and methods .

Recall that the method getTime():long of type Date returns the number of milliseconds that corresponds to the given Date object.

TripPoint

```
public ______________________________________________
private int id;
private Location location;
private long boardingTime;
public TripPoint(int id, Location location, long boardingTime) {
  this.id = id;
  this.location = location;
  this.boardingTime = boardingTime;
}
public int getId() {
  return id;
}
public Location getLocation() {
  return location;
}
public long getBoardingTime() {
  return boardingTime;
}
@Override
public boolean equals(Object other) {
  return other != null
  && other.getClass().equals(getClass())
  && id == ((TripPoint)other).getId();
}
```

TrainStation

```
public ________________________________________________
public TrainStation(int id, Location location, long boardingTime) {
}
}
Airport
public ________________________________________________
private double airportTax;
private long minLayover;
public Airport(int id, Location location, double airportTax, long boardingTime, long
    minLayover) {
}
public double getAirportTax() {
  return airportTax;
}
public long getMinLayover() {
  return minLayover;
}
}
```

Trip

```
public ________________________________________________
private TripPoint source;
private TripPoint destination;
private Date departure;
private Date arrival;
private double basePrice;
public Trip(TripPoint source, TripPoint destination, Date departure, Date arrival, double
    basePrice) {
  this.source = source;
  this.destination = destination;
  this.departure = departure;
  this.arrival = arrival;
  this.basePrice = basePrice;
}
public TripPoint getSource() {
  return source;
}
public TripPoint getDestination(){
  return destination;
}
public Date getDeparture() {
  return departure;
}
public Date getArrival() {
  return arrival;
}
public double getBasePrice() {
  return basePrice;
}
```

Flight

```
public ________________________________________
public Flight(Airport source, Airport destination, Date departure, Date arrival, double
    basePrice) {
  super(source, destination, departure, arrival, basePrice);
}
}
```

Train

```
public ________________________________________
public Train(TrainStation source, TrainStation destination, Date departure, Date arrival,
    double basePrice) {
  super(source, destination, departure, arrival, basePrice);
}
}
```

TripPlanner

```java
public ______________________________________________
/**
* Returns whether the sequence of Trips first and second is vald.
*
* @param first the first Trip in the two-trip sequence
* @param second the second Trip in the two-trip sequence
* @param maxLayover the maximum time between the two Trips, so that the sequence is considered
* valid
* @return true,
* if the arrival point of the first Trip is the same as the departure point of the
* second Trip,
* if the layover between the two Trips is no bigger than maxLayover, and
* if the layover between the two Trips is large enough, according to the system description,
* and false, otherwise.
*
*/
public static boolean isValid(Trip first, Trip second, long maxLayover) {
//TODO: please implement this
}
/**
* Returns the total cost of the given given sequence of trips.
* @param trips the sequence to calculate the cost
* @return the total cost of trips
*/
public static double cost(List<Trip> trips) {
//TODO: please implement this
}

/**
* Returns whether the sequence of Trips is valid.
* @param trips the sequence of Trips to examine for validity
* @param maxLayover the maximum time between the two Trips, so that the sequence is considered
* valid
* @return true, if every consecutive pair in the given sequence is valid.
* An empty sequence is considered valid.
*/
public static boolean isValid(List<Trip> trips, long maxLayover) {
//TODO: please implement this
}
/**
* Returns the total time of the given given sequence of trips.
* @param trips the sequence to calculate the time
* @return the total time of trips
*/
public static long time(List<Trip> trips) {
//TODO: please implement this
}
```

Question 8: Open Question— Program Slicing (10 points)

Program slicing is a technique for automatically simplifying programs by omitting irrelevant operations. Slicing reduces a program to a minimal form which preserves the behaviour of a selected statement. Program slicing can be used in debugging to locate source of errors more easily by ruling out statements that do not contribute to a faulty behaviour. In this question, you will apply program slicing to analyze a real bug in Mozilla.

An example of program slicing is as follows:

```
int i;
int sum = 0;
int product = 1;
for(i = 1; i < N; ++i) {
  sum = sum + i;
  product = product * i;
  }
write(sum);
write(product);
```

We can slice this program with respect to the statement (**write(sum)**), thus omitting, for instance, all calculations of the irrelevant product variable. A slice of a program with respect to a statement **s** must include all statements which (transitively) define values that are used in **s** (that is, all statements that contribute data flow to **s**). Furthermore, a slice must also include all control-flow statements **s0** which affect whether or not **s** is reachable, along with all statements that contribute data flow to **s0**.

For **write(sum)**, we keep the define of **sum** variable in the loop; that define also uses the initial define of **sum** before the loop.

```
int i;
int sum = 0;

for(i = 1; i < N; ++i) {
  sum = sum + i;

  }
write(sum);
```

The Mozilla bug reported in its bug report 192226 was caused by the code below. The conditional expression in the second if block (Line 1923) leads to a Null Pointer Exception at "child = child.getNext().getNext();" (Line 1927).

```
1876 private void visitRegularCall(Node node, int type,
Node child, boolean firstArgDone)
1878{
...
1895 int childCount = 0;
1896 int argSkipCount = (type == TokenStream.NEW) ? 1 : 2;
1897 while (firstArgDone && (child != null)) {
1898   childCount++;
1899   child = child.getNext();
1900 }
...
1905 int argIndex = -argSkipCount;
1906 if (firstArgDone && (child != null)) {
1907   child = child.getNext();
1908   argIndex++;
1909   aload(contextLocal);
1910   addByteCode(ByteCode.SWAP);
1911 }
...
1918 boolean isSpecialCall = node.getProp(Node.SPECIALCALL_PROP) != null;
1919 boolean isSimpleCall = false;
1920 String simpleCallName = null;
...
1922 simpleCallName = getSimpleCallName(node);
1923 if (simpleCallName != null && !isSpecialCall) {
1924   isSimpleCall = true;
1925   push(simpleCallName);
1926   aload(variableObjectLocal);
1927   child = child.getNext().getNext();
1928   argIndex = 0;
1929   push(childCount - argSkipCount);
1930   addByteCode(ByteCode.ANEWARRAY, "java/lang/Object");
1931 }
...
2053}
```

Give the program slicing of method **visitRegularCall(...)**, with respect to the statement **(child = child.getNext().getNext())** You may indicate the slice by writing down a sequence of line numbers, or by copying the lines themselves along with the line numbers. [10 points]

Formula Sheet for Design by Contract

JML Syntax:

1. Pre-condition: //requires formula
2. Post-condition: //ensures formula
3. Loop Invariant/variant: //loop_invariant formula
4. Class Invariants //invariant formula

Expressions:

1. \old(E): defined to be value of E in pre-state
2. \result: defined to be result of method call

Logical expressions: ==>, <==, <==>, <=! =>

Quantifiers: \forall, \exists