Exercise Set: Data Abstraction

In this exercise set, we have marked questions we think are harder than others with a **[‡]**. We have also marked questions for which solutions are provided at the end of the set (**[SP]**). To check solutions for other questions than those marked with **[SP]**, ask one of the instructors or TAs or post a question to Piazza!

1. Answer the following questions with true or false and explain your choice in one sentence. **[SP]**

a) In an object-oriented programming language, data abstraction is mainly used to describe the details of how the data is represented.

b) The MODIFIES clause is used to describe how the method modifies the object and any parameters to the method.

c) The following code snippet has a side effect.

Note: the statement `currentStorageFormat.getFileFilter().accept(file)` demonstrates how in Java you can link method calls together. On the object referenced by the `currentStorageFormat` variable, call the method `getFileFilter` and on the result of that method call (which must be an object returned) call the `accept` method.

```
public StorageFormat findStorageFormat(File file) {
    StorageFormat currentStorageFormat;
     for (int i=0; i<myStorageFormats.size(); i++) {
         currentStorageFormat = (StorageFormat) myStorageFormats.get(i);
         if (currentStorageFormat.getFileFilter().accept(file)) {
             return currentStorageFormat;
         }
    }
     return null;
}
```

d) A method is an example of the use of procedural abstraction.

e) The following code snippet has a side effect.

```
public void setVoteCount(int voteCount) {
     this.voteCount = voteCount;
}
```

f) The following code snippet has a side effect.

```
public boolean hasTenMoreElements(List<Object> elements) {
     int size = elements.size();
     size = size - 10;
     return (size == 0);
}
```

g) Declaring fields that capture the state of an object as **`private`** supports good data abstraction.

h) [Not examinable but good practice!] The `IntegerSet` class from the lecture is immutable. **[‡]**

i) A black-box test can be written without any knowledge about the details of a concrete implementation, the specification of the data abstraction is enough.

j) Assuming List refers to the `List` defined in the Java Standard Libraries, the following code is correct.

```
List<String> l = new List<String>();
```

2. The following method definition was shown in the reading. From its specification, do you know whether or not the method has a side effect and why or why not? Explain briefly. **[SP]**

```
// Requires: foodEaten must be a non-empty list
// Modifies: this and foodEaten
// Effects: this remembers foodEaten and foodEaten is empty
void recordLastFeeding(List<FeedingRecord> foodEaten) {...}
```

3. If a method changes member variables of a class, why do you not specify the member variables being changed by the method in the effects clause, but rather only state that "this" is being changed?

4. Given the method `mirrorInputString` below:

a) Write the MODIFIES and EFFECTS clause for the method, if any, based on the code shown.

Note: A `StringBuffer` is similar to a `String`, but can be modified. The length of a `StringBuffer` can be changed, for example, by appending a `String` at the end of the buffer using the method `append`. The method `toString` of `StringBuffer` returns a `String` that represents the data in the `StringBuffer`.

```
public String mirrorInputString(String s) {
    StringBuffer mirroredStringBuffer = new StringBuffer();
    for (int i=s.length()-1; i>=0; i--) {
        mirroredStringBuffer.append(s.charAt(i));
    }
    String mirroredString = mirroredStringBuffer.toString();
    return mirroredString;
}
```

5. Does the following method mutate the object used to invoke it or simply access information about the object? (Explain briefly.) **[‡]**

```
// Modifies: user
// Effects: the name of the user is changed to newName
public void changeUserName(User user, String newName) {
    user.setName(newName);
}
```

6. Given the following `EllipseFigure` class from JHotdraw below (some parts are left out for ease of use): **[‡]**, **[SP]**
a) Specify the REQUIRES, MODIFIES and EFFECTS clauses for each method as best you can based on the code.
b) Which is/are the constructor(s) in this class?
c) Specify for each method whether it is a mutator (i.e., it modifies `this`) or an observer (i.e., it does not modify `this`) method.
d) Is the class itself mutable or immutable? (Explain in one sentence why.)

```
/**
 * An ellipse figure.
 */
public class EllipseFigure extends AttributeFigure {
    private Rectangle fDisplayBox;

    public EllipseFigure() {
        this(new Point(0,0), new Point(0,0));
    }

    public EllipseFigure(Point origin, Point corner) {
        basicDisplayBox(origin,corner);
    }

    public void basicDisplayBox(Point origin, Point corner) {
        this.fDisplayBox = new Rectangle(origin);
        this.fDisplayBox.add(corner);
    }

    public Rectangle displayBox() {
        return new Rectangle(
            fDisplayBox.x,
            fDisplayBox.y,
            fDisplayBox.width,
            fDisplayBox.height);
    }

    public void basicMoveBy(int x, int y) {
        fDisplayBox.translate(x,y);
    }

    public void drawBackground(Graphics g) {
        Rectangle r = displayBox();
        g.fillOval(r.x, r.y, r.width, r.height);
    }
```

```
    public void drawFrame(Graphics g) {
        Rectangle r = displayBox();
        g.drawOval(r.x, r.y, r.width-1, r.height-1);
    }

    public Insets connectionInsets() {
        Rectangle r = fDisplayBox;
        int cx = r.width/2;
        int cy = r.height/2;
        return new Insets(cy, cx, cy, cx);
    }

    public void write(StorableOutput dw) {
        dw.writeInt(fDisplayBox.x);
        dw.writeInt(fDisplayBox.y);
        dw.writeInt(fDisplayBox.width);
        dw.writeInt(fDisplayBox.height);
    }

    public void read(StorableInput dr) throws IOException {
        fDisplayBox = new Rectangle(
            dr.readInt(),
            dr.readInt(),
            dr.readInt(),
            dr.readInt());
    }
}
```

7. Given the partial `LineFigure` class definition below:
a) Which fields would you define for this class and why? (Try to come up with a simple and expressive way to specify the data.) **[‡]**
b) Is the class mutable or immutable? (Explain in one sentence why.) **[‡]**
c) What would be good black-box test cases for the method `moveLineBy` and why?

```
public class LineFigure {
    // construct a line
    // Effects: this updated with the given coordinates
    public LineFigure(int x1, int y1, int x2, int y2) {...}

    // Modifies: this
    // Effects: this is moved by dx and dy
    public void moveLineBy(int dx, int dy) {...}

    // Modifies: this
    // Effects: the color of this is updated to the one
      // specified by the rgb values
    public void setColor(int r, int g, int b) {...}

    // Modifies: g
    // Effects: draws the line onto g
    public void drawLine(Graphics g) {...}
    ...

}
```

8. Why are formal reasoning techniques not applicable for most data abstractions? **[‡]**

9. Given the specification of the following method in a `Line` class; what would be good test cases? **[SP]**

```
// Requires: dx and dy are positive and less or equal to 100
// Modifies: this
// Effects: this is moved by dx and dy
public void moveLineBy(int dx, int dy) {...}
```

10. Given a method that takes a `String` as input and returns a `String` that is the mirrored representation of the input `String` with the following partial specification: What would be good test cases?

```
// Requires: String s is at most 10 characters long
public String mirrorInputString(String s) {...}
```

11. Why is it often difficult to use Java classes from other software in your own software? What can you do to better understand the functionality of a Java class written by someone else? **[‡]**

12. The reading talks about the `List` data abstraction from the Java Collections Framework. **[‡]**
a) Do you think that the `List` data abstraction specifies the fields of the data abstraction and why or why not?
b) Do you think that a `List` data abstraction, like the one from the Java Collections Framework (JCF), is mutable or immutable and why?
c) Why do you think the `List` data abstraction was introduced instead of just having a data abstraction for `ArrayList` and `LinkedList`?

13. Let's pretend we are building a discussion group system like piazza that we use in the course. Such a system would need some way to represent a note that can be posted to a course. In discussions with potential users, you might determine a "Note" would need to support the following functionality:

- A title for the note
- The contents of the note
- An ability to edit the contents of the note
- An ability to attach one file to a note
- An ability to start a follow-up discussion

`(a)` Here is an outline of the class definition you might start with to represent a Note in this system. Fill in suitable specifications for the Constructor and methods of this class. You can assume that editing of a note will be provided by the user interface that will get the contents of the note, allow the contents to be changed and then will set the contents back into the Note after editing is complete. **[SP]**

```
public class Note {
   private String title;
   private TextBlock contents;
   private FollowUp noteFollowUp;
   private File fileForNote;

   // Constructor to create an empty new note
  // FILL IN SPECIFICATION
   public Note()  {}

   //  Set the title of the note
   // FILL IN SPECIFICATION
   public void setTitle(String title) {}

   // Set the contents of the note
   // FILL IN SPECIFICATION
   public void setContents(TextBlock contents) {}

   // Get the contents of the note
   // FILL IN SPECIFICATION
   public TextBlock getContents() {}

   // Attach a file to the note
   // FILL IN SPECIFICATION
   public void attachFile(File fileToAttach) {}

   // Start a follow-up discussion
   // FILL IN SPECIFICATION
   public void initiateFollowUp() {}

}
```

(b) Starting with the code below, sketch the Java statements (don't worry too much about syntax) to set the title of aNote to "new note" and to start a follow-up discussion. **[SP]** Sketch the code to also attach a file. **[‡]**

```
Note aNote = new Note();
```

(c) Define three tests for Note. (There are more tests to be written, but let's not belabor it here! Try on your own…) **[‡]**

(d) Implement the Note class **[‡]**

14. [Note, focus on the weekly programming assignments first!] In this question we ask you to specify, use, test and implement a new data abstraction. Suppose you are working with a meteorologist who is interested in collecting and analyzing rainfall data. Each record of rainfall is measured to the nearest millimetre. We want to be able to add a record or a list of records to the rainfall data, get the number of records entered, get the total rainfall over all records entered and get the largest of the rainfall records entered. We assume that there is always at least one rainfall record – consequently, your constructor must take a parameter that represents the first rainfall record. [**[‡],SP]**

a) In IntelliJ, start a new project and create a package named `ca.ubc.cpsc210.rainfall`. Create a new class named `RainfallRecords`

b) Complete the *specification* of the `RainfallRecords` data type

c) Now we will *use* this new data type. Sketch out the code to perform the following (do this on paper, not in IntelliJ):

- create a new `RainfallRecords` object with 0 as the first record entered
- add the following records: 4, 5, 0, 0, 0, 2, 3
- determine the total rainfall, maximum rainfall and number of records entered and assign each of these to a new variable of the appropriate type

d) In IntelliJ, open the `RainfallRecords` class in the editor window. Click the name of the class and wait for a yellow light-bulb to appear. Click the light-bulb and select **Create Test**. Select **JUnit4**. You will see a warning that the JUnit4 library was not found in the module, click **Fix**. Use `RainfallRecordsTest` for the name of the test class. For the destination package enter `ca.ubc.cpsc210.rainfall.test`. Design unit tests for the `RainfallRecords` data type.

e) In IntelliJ, write the implementation of the `RainfallRecords` constructor and methods. Run the tests and debug until all tests pass.

Note: there is more than one viable implementation for this class!

SOLUTIONS:

1.
a) False.
Data abstraction is mainly used to describe how a data object behaves rather than how the data is represented.

b) False.
The MODIFIES clause states which input values are modified by the method, not how the values are modified.

c) False.
The method does not modify a field of the class.

d) True.
The method abstracts from the sequence of steps to be done to perform an action.

e) True.
The method modifies the implicit input value this.

f) False.
No input value is being modified.

g) True.
Fields capturing the state of an object should not be accessed directly. They should only be accessed by operations of the data abstraction.

h) False.
Objects of type `IntSet` have state that changes as operations are executed (elements can be added and removed).

i) True.
A black-box test exercises the implementation in ways the abstraction has specified to support without making any assumptions about what the implementation may allow.

j) False.
`List` is a Java interface; as interfaces cannot include the definitions of constructors, this code is incorrect.

2. The method has a side effect as `this` is being modified by the operation so that it remembers `foodEaten`, and the change persists after the operation executed.

6. a)

```
// Effects: the position of the figure stored in this is set
public EllipseFigure() {
    this(new Point(0,0), new Point(0,0));
}

// Effects: this is updated with origin and corner
public EllipseFigure(Point origin, Point corner) {
    basicDisplayBox(origin,corner);
}


// Modifies: this
// Effects: this is updated with origin and corner
public void basicDisplayBox(Point origin, Point corner) {
    this.fDisplayBox = new Rectangle(origin);
    this.fDisplayBox.add(corner);
}

// Effects: returns the display box (a rectangle) of the ellipse
public Rectangle displayBox() {
    return new Rectangle(
        fDisplayBox.x,
        fDisplayBox.y,
        fDisplayBox.width,
        fDisplayBox.height);
}

// Modifies: this
// Effects: this is moved by x and y
public void basicMoveBy(int x, int y) {
    fDisplayBox.translate(x,y);
}

// Modifies: g
// Effects: the ellipse's background is drawn onto g
public void drawBackground(Graphics g) {
    Rectangle r = displayBox();
    g.fillOval(r.x, r.y, r.width, r.height);
}

// Modifies: g
// Effects: the ellipse's frame is drawn onto g
public void drawFrame(Graphics g) {
    Rectangle r = displayBox();
    g.drawOval(r.x, r.y, r.width-1, r.height-1);
}

// Effects: returns connection insets for the ellipse, i.e. how much space
has to be left as border (this is not really visible from the code!)
public Insets connectionInsets() {
    Rectangle r = fDisplayBox;
    int cx = r.width/2;
    int cy = r.height/2;
    return new Insets(cy, cx, cy, cx);
}
```

```
// Modifies: dw
// Effects: writes the ellipse's data into storable output
public void write(StorableOutput dw) {
    dw.writeInt(fDisplayBox.x);
    dw.writeInt(fDisplayBox.y);
    dw.writeInt(fDisplayBox.width);
    dw.writeInt(fDisplayBox.height);
}

// Modifies: this
// Effects: reads the ellipse's data from storable input
public void read(StorableInput dr) throws IOException {
    fDisplayBox = new Rectangle(
        dr.readInt(),
        dr.readInt(),
        dr.readInt(),
        dr.readInt());
}
```

b) The constructors in the class are:

```
public EllipseFigure() {...}
```

and

```
public EllipseFigure(Point origin, Point corner) {...}
```

c) Mutators: `basicDisplayBox, basicMoveBy, read`
Observers: `displayBox, drawBackground, drawFrame, connectionInsets, write`

d) The class `EllipseFigure` is mutable as it has methods that modify `this`.

9. Good test cases test typical input values as well as values at the boundaries defined by the input values data type and the REQUIRES clause. In our example, a good test case should create a `Line` object and then test typical input values such as (50,50) for (dx,dy), and then also test the boundaries: (1,1) as the values are positive, and (100,100) as the values for (dx,dy) are less or equal to 100.

13 a. One possible answer of several…

```
public class Note {
   private String title;
   private TextBlock contents;
   private FollowUp noteFollowUp;
   private File fileForNote;

   // Constructor to create an empty new note
   // Requires: nothing
   // Modifies: this
   // Effects: a note with an empty title and contents is created
   public Note()  {}

   //  Set the title of the note
   // Requires: title is a non-empty String
   // Modifies: this
   // Effects:  this.title = title
   public void setTitle(String title) {}

   // Set the contents of the note
   // Requires:  contents is a non-empty text block
   // Modifies: this
   // Effects: this.contents = contents
   public void setContents(TextBlock contents) {}

   // Get the contents of the note
   // Requires: nothing
   // Modifies: nothing
   // Effects: returns contents TextBlock, may be empty
   public TextBlock getContents() {}

   // Attach a file to the note
   // Requires: fileToAttach is well-formed file available on filesystem
   // Modifies: this
   // Effects this.fileForNote = fileToAttach
   public void attachFile(File fileToAttach) {}

   // Start a follow-up discussion
   // Requires: nothing
   // Modifies: this
   // Effects: this.noteFollowUp is a new follow-up that overwrites a
   // follow-up that already exists
   public void initiateFollowUp() {}

}
```

14. Will be made available through the SVN \lectures repository as a project named RainFallData