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 the google group!

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) The IntSet 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) Is the method a mutator or an observer method?
b) 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=0; i<s.length(); i++) {
            mirroredStringBuffer.append(s.indexOf(i));
      }
      String mirroredString = mirroredStringBuffer.toString();
      return mirroredString;
}
```

5. Is the following method a mutator or an observer method? (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 or an observer 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 Line 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, as the one from the JCF, 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?

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 can not 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.