Com S 362
Fall 2002

Name: _____

Object-Oriented Analysis and Design

Exam 1 on Abstract Data Types and Java

This test has 3 questions and pages numbered 1 through 13.

Reminders

This test is open book and notes. However, it is to be done individually and you are not to exchange or share materials with other students during the test.

If you need more space, use the back of a page. Note when you do that on the front.

This test is timed. We will not grade your test if you try to take more than the time allowed. Therefore, before you begin, please take a moment to look over the entire test so that you can budget your time.

For diagrams and programs, clarity is important; if your diagrams or programs are sloppy and hard to read, you will lose points. Correct syntax also makes some difference.

---

1. (10 points) Name one feature of Java that we have studied that, when used, makes programs easier to change. Explain how it helps in a brief sentence.

2. This is a problem about simple applications, file I/O, and exceptions in Java. In this problem you will write your code by filling in the empty methods in the code below. The program consists of 3 classes, found below. The details are in the program's comments. (An overview can be found in part (c) below, which contains the main class and the main method.)

   (a) (5 points) In the following class, fill in the true branch of the `if` statement in the body of the method "`readEmailAddr`".

```
package cs362exams;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.LineNumberReader;
import java.io.Reader;

/** Text based user interface for getting email addresses.
 */
public class EmailDialogUI {

    /** The input. */
    private BufferedReader in
       = new BufferedReader(new InputStreamReader(System.in));

    /**
     * Read a line from System.in and return it.
     * If the line read is null (which means an end of file is reached),
     * then throw an IOException with a message indicating the problem.
     * The message reads:
     *    "End of file while reading email address"
     * @throws IOException if end of file is reached
     */
    //@ ensures \result != null;
    public String readEmailAddr() throws IOException {
        String addr = in.readLine();
        if (addr == null) {


        }
        return addr;
    }

    /**
     * Prompt the user to enter their email address by writing
     *    "Enter your email address: "
     * to System.out.  There should be no newline
     * at the end of the prompt.
```

```
     */
    public void promptForEmailAddr() {
        System.out.print("Enter your email address: ");
    }

    /**
     * Tell the user that inconsistent addresses were entered by
     * a message of the form:
     *    "The addresses ‘<firstaddr>’ and ‘<secondaddr>’ don’t match"
     * to System.err, followed by a newline.
     * @param addr1 the first address entered (i.e., <firstaddr>)
     * @param addr2 the second address entered (i.e., <secondaddr>)
     */
    //@ requires addr1 != null && addr2 != null;
    //@ requires !(addr1.equals(addr2));
    public void tellAboutAddressInconsistency(String addr1, String addr2) {
        System.err.println("The addresses ‘" + addr1
                           + "’ and ‘" + addr2
                           + "’ don’t match.");
    }
}
```

(b) (5 points) In the following class, "`EmailDialog`," fill in the body of the constructor.

(c) (15 points) Also in the following class, fill in the body of the method "`writeIntoFile`".

```
package cs362exams;

import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;

/** The application logic for the email address dialog.
 */
public class EmailDialog {

    /** The user interface for this dialog. */
    private EmailDialogUI ui;

    /** Initialize this dialog, so that it talks to the given UI. */
    public EmailDialog(EmailDialogUI ui) {



    }

    /** Run the dialog.
     *  That is, prompt the user on standard output,
     *  then read the email address on a line from standard input,
     *  then repeat, then compare the two addresses.
     *  If they are the same, then write the address to the file
     *  "emails.txt" and return true.  Otherwise, inform the user
     *  through the user interface and return false.
     *  @return true if and only if the two addresses read were the same
     *  @throws IOException if I/O problems occur
     */
    public boolean doIt() throws IOException {
        String addr1 = promptAndRead();
        String addr2 = promptAndRead();
        if (addr1.equals(addr2)) {
            writeIntoFile(addr1);
            return true;
        } else {
            ui.tellAboutAddressInconsistency(addr1, addr2);
            return false;
        }
    }

    /**
     * Prompt the user to enter their address, read it in, and return it.
```

```
     * @return String the email address
     */
    private String promptAndRead() throws IOException {
        ui.promptForEmailAddr();
        return ui.readEmailAddr();
    }

    /**
     * Write the given email address into the end of the emails.txt file.
     * You can use the expression
     *      new PrintWriter(new FileWriter(outFile, true))
     * to create a PrintWriter that will output to the end of the
     * File outFile.  We expect that you know how to create a File object
     * from a String so that it is open on the file named "emails.txt".
     * Be sure to close the file when you are done.
     * @param addr1 the email address.
     * @throws IOException if any I/O problems arise
     */
    private void writeIntoFile(String addr1) throws IOException {









    }
}
```

(d) (10 points) In the following class, fill in the body of the method "`main`".

```
package cs362exams;

import java.io.IOException;

/** Get a user's email address and record it in a file.
 *  The user is prompted on standard output, and the email address is read.
 *  This is repeated. If the addresses entered are the same,
 *  then write it to the end of the file emails.txt (followed by a newline).
 *  Otherwise, print an error message and exit with a non-zero error code.
 */
public class EmailDialogMain {

    /** Run this application.  This method is responsible for calling
     *  the doIt method of the class EmailDialog.  If that method
     *  returns false, then this method exits (by calling System.exit)
     *  with a non-zero exit code.  This method is also responsible for
     *  catching any IOExceptions that the doIt method may throw,
     *  and if it catches them it should print their message to System.err,
     *  preceeded by "EmailDialogMain: I/O error: " and followed by a newline.
     * @param args is ignored.
     */
    public static void main(String[] args) {









    }
}
```

3. (50 points)

This is a problem about writing a correct implemetation of an ADT. You are to fill in the body of the methods in the class below, and also declare its fields. Be sure to declare these fields with the appropriate privacy. Following the class is a JUnit test class for it. You can refer to this JUnit test class as a supplement to the comments that specify the class.

```
package cs362exams;

//@ model import org.jmlspecs.models.JMLDouble;

/** Spaceships for a (two dimensional) computer game.
 * The spaceship starts out at a position specified by the constructor,
 * with a heading of 0 radians (straight out along the positive x axis),
 * and initially it is at rest (no velocity or acceleration).
 * The spaceship can be moved by changing its heading and giving it some
 * acceleration.  The spaceship only has one thrust engine at its rear,
 * so it can only accelerate along its heading.  It cannot use
 * negative acceleration, but must turn around to slow down.
 * The ship uses Newtonian physics and ignores relativistic effects.  */
public class SpaceShip {     // DECLARE ANY FIELDS YOU MAY NEED BELOW









    /**
     * Initialize this SpaceShip to be at the given coordinates,
     * with heading angle 0, with velocity 0, and with acceleration 0.
     */
    public SpaceShip(double x, double y) {



    }

    /** Returns this ship's x coordinate. */
    public /*@ pure @*/ double getX() {


    }
```

```
/** Returns this ship's y coordinate. */
public /*@ pure @*/ double getY() {


}

/** Return this ship's heading angle.
 * The angle is measured, counterclockwise from the positive x axis,
 * in radians.
 */
//@ ensures 0 <= \result && \result < 2*Math.PI;
public /*@ pure @*/ double getHeadingAngle() {


}

/** Returns the x component of this ship's velocity. */
public /*@ pure @*/ double getXVelocity() {


}

/** Returns the y component of this ship's velocity. */
public /*@ pure @*/ double getYVelocity() {


}

/** Returns this ship's (average) acceleration (along it's heading). */
public /*@ pure @*/ double getAcceleration() {


}

/** Set this ship's heading to be the given angle,
 *  measured in radians, counterclockwise from the positive
 *  x axis.  Setting the heading has no effect on the acceleration.
 *  @param angle The heading angle desired     */
//@ requires 0 <= angle && angle < 2*Math.PI;
//@ ensures angle == getHeadingAngle();
//@ ensures getX() == \old(getX()) && getY() == \old(getY());
//@ ensures getXVelocity() == \old(getXVelocity());
//@ ensures getYVelocity() == \old(getYVelocity());
//@ ensures getAcceleration() == \old(getAcceleration());
public void setHeadingAngle(double angle) {


}
```

```
/** Set this ship's (average) acceleration along its heading
 *  to be the given value.
 *  @param acceleration The desired acceleration.
 */
//@ requires acceleration >= 0.0;
//@ ensures getAcceleration() == acceleration;
//@ ensures getX() == \old(getX()) && getY() == \old(getY());
//@ ensures getHeadingAngle() == \old(getHeadingAngle());
//@ ensures getXVelocity() == \old(getXVelocity());
//@ ensures getYVelocity() == \old(getYVelocity());
public void setAcceleration(double acceleration) {


}

/** The tolerance for sloppiness in floating point calculations.
 */
public static final double tolerance = 0.001;

/** Update the position and velocity of this ship,
 *  after the passing of the given number of time units.
 *  The heading and acceleration are unchanged.
 *  @param timeUnits - units of time that have passed.
 */
//@ requires timeUnits > 0;
/*@ ensures JMLDouble.withinEpsilonOf(
  @             \old(getX()
  @                  + (0.5 * getAcceleration()
  @                         * Math.cos(getHeadingAngle())
  @                         * timeUnits * timeUnits)
  @                  + getXVelocity() * timeUnits),
  @             getX(),
  @             tolerance);
  @ ensures JMLDouble.withinEpsilonOf(
  @             \old(getY()
  @                  + (0.5 * getAcceleration()
  @                         * Math.sin(getHeadingAngle())
  @                         * timeUnits * timeUnits)
  @                  + getYVelocity() * timeUnits),
  @             getY(),
  @             tolerance);
  @ ensures JMLDouble.withinEpsilonOf(
  @             \old(getXVelocity()
  @                  + getAcceleration() * Math.cos(getHeadingAngle())
  @                                      * timeUnits),
  @             getXVelocity(),
  @             tolerance);
```

```
    @ ensures JMLDouble.withinEpsilonOf(
    @             \old(getYVelocity()
    @                  + getAcceleration() * Math.sin(getHeadingAngle())
    @                                      * timeUnits),
    @             getYVelocity(),
    @             tolerance);
    @ ensures getHeadingAngle() == \old(getHeadingAngle());
    @ ensures getAcceleration() == \old(getAcceleration());
    @*/
  public void update(int timeUnits) {

  }
}
```

The following is a JUnit test class for the class `SpaceShip`. You don't have to read this if you understand what to do already.

```
package cs362exams;

import junit.framework.TestCase;

/** Test for the SpaceShip class.
 */
public class SpaceShipTest extends TestCase {

    /** Initialize this SpaceShipTest. */
    public SpaceShipTest(String name) {
        super(name);
    }

    /** Run the test cases in this class. */
    public static void main(String[] args) {
        junit.textui.TestRunner.run(SpaceShipTest.class);
    }

    private static double tolerance = 0.001;

    /** Test for the SpaceShip constructor. */
    public void testSpaceShip() {
        SpaceShip s = new SpaceShip(3.14, 2.73);
        SpaceShip s2 = new SpaceShip(-5.0, 62.0);
        assertEquals(3.14, s.getX(), tolerance);
        assertEquals(2.73, s.getY(), tolerance);
        assertEquals(0.0, s.getHeadingAngle(), tolerance);
        assertEquals(0.0, s.getXVelocity(), tolerance);
        assertEquals(0.0, s.getYVelocity(), tolerance);
        assertEquals(0.0, s.getAcceleration(), tolerance);
        assertEquals(-5.0, s2.getX(), tolerance);
        assertEquals(62.0, s2.getY(), tolerance);
        assertEquals(0.0, s2.getHeadingAngle(), tolerance);
        assertEquals(0.0, s2.getXVelocity(), tolerance);
        assertEquals(0.0, s2.getYVelocity(), tolerance);
        assertEquals(0.0, s2.getAcceleration(), tolerance);
    }


    /** Test for the setHeadingAngle method. */
    public void testSetHeadingAngle() {
        SpaceShip s = new SpaceShip(5.0, 7.0);
        SpaceShip s2 = new SpaceShip(36.0, 2.1);
        s.setHeadingAngle(2.0);
        assertEquals(2.0, s.getHeadingAngle(), tolerance);
```

```
    s2.setHeadingAngle(1.0);
    assertEquals(1.0, s2.getHeadingAngle(), tolerance);
    // see that nothing else changed
    assertEquals(5.0, s.getX(), tolerance);
    assertEquals(7.0, s.getY(), tolerance);
    assertEquals(0.0, s.getXVelocity(), tolerance);
    assertEquals(0.0, s.getYVelocity(), tolerance);
    assertEquals(0.0, s.getAcceleration(), tolerance);
    assertEquals(36.0, s2.getX(), tolerance);
    assertEquals(2.1, s2.getY(), tolerance);
    assertEquals(0.0, s2.getXVelocity(), tolerance);
    assertEquals(0.0, s2.getYVelocity(), tolerance);
    assertEquals(0.0, s2.getAcceleration(), tolerance);
    assertEquals(2.0, s.getHeadingAngle(), tolerance);
}

/** Test for the setAcceleration method. */
public void testSetAcceleration() {
    SpaceShip s = new SpaceShip(5.0, 7.0);
    SpaceShip s2 = new SpaceShip(36.0, 2.1);
    s.setAcceleration(17.0);
    assertEquals(17.0, s.getAcceleration(), tolerance);
    s2.setAcceleration(21.0);
    assertEquals(21.0, s2.getAcceleration(), tolerance);
    // see that nothing else changed
    assertEquals(0.0, s.getHeadingAngle(), tolerance);
    assertEquals(5.0, s.getX(), tolerance);
    assertEquals(7.0, s.getY(), tolerance);
    assertEquals(0.0, s.getXVelocity(), tolerance);
    assertEquals(0.0, s.getYVelocity(), tolerance);
    assertEquals(0.0, s2.getHeadingAngle(), tolerance);
    assertEquals(36.0, s2.getX(), tolerance);
    assertEquals(2.1, s2.getY(), tolerance);
    assertEquals(0.0, s2.getXVelocity(), tolerance);
    assertEquals(0.0, s2.getYVelocity(), tolerance);
}

/** Test for the update method. */
public void testUpdate() {
    SpaceShip s = new SpaceShip(0.0, 0.0);
    SpaceShip s2 = new SpaceShip(-10.0, -10.0);
    s.setAcceleration(1.0);
    s2.setAcceleration(10.0);
    s.setHeadingAngle(Math.PI/2);
    s2.setHeadingAngle(Math.PI/4);
    s.update(1);
    assertEquals(0.0, s.getX(), tolerance);
    assertEquals(0.5*1*1*Math.sin(Math.PI/2), s.getY(), tolerance);
```

```
        assertEquals(0.0, s.getXVelocity(), tolerance);
        assertEquals(1*1*Math.sin(Math.PI/2), s.getYVelocity(), tolerance);
        assertEquals(1.0, s.getAcceleration(), tolerance);
        assertEquals(Math.PI/2, s.getHeadingAngle(), tolerance);
        s.update(2);
        assertEquals(0.0, s.getX(), tolerance);
        assertEquals(0.5 + 0.5*2*2*Math.sin(Math.PI/2) + 2*1.0,
                     s.getY(), tolerance);
        assertEquals(0.0, s.getXVelocity(), tolerance);
        assertEquals(3.0, s.getYVelocity(), tolerance);
        assertEquals(1.0, s.getAcceleration(), tolerance);
        assertEquals(Math.PI/2, s.getHeadingAngle(), tolerance);
        s2.update(3);
        assertEquals(-10.0 + 0.5*3*3*10*Math.cos(Math.PI/4),
                     s2.getX(), tolerance);
        assertEquals(-10.0 + 0.5*3*3*10*Math.sin(Math.PI/4),
                     s2.getY(), tolerance);
        assertEquals(3*10*Math.cos(Math.PI/4), s2.getXVelocity(), tolerance);
        assertEquals(3*10*Math.sin(Math.PI/4), s2.getYVelocity(), tolerance);
        assertEquals(10.0, s2.getAcceleration(), tolerance);
        assertEquals(Math.PI/4, s2.getHeadingAngle(), tolerance);
    }
}
```