CS 2510 Exam 1 – Summer 2010

Name: _____

Student Id (last 4 digits): _____

• Write down the answers in the space provided.
• You may use all syntax that you know from *FunJava* other than *abstract classes*. If you need a method and you don't know whether it is provided, define it. You do not need to include the curly braces for every `if` or every `else`, as long as the statements you write are otherwise correct in FunJava.
• For tests you only need to provide the expression that computes the actual value, connecting it with an arrow to the expected value. For example `s.method() -> true` is sufficient.
• Remember that the phrase "design a class" or "design a method" means more than just providing a definition. It means to design them according to the **design recipe**. You are *not* required to provide a method template unless the problem specifically asks for one. However, be prepared to struggle if you choose to skip the template step.
• We will not answer *any* questions during the exam.

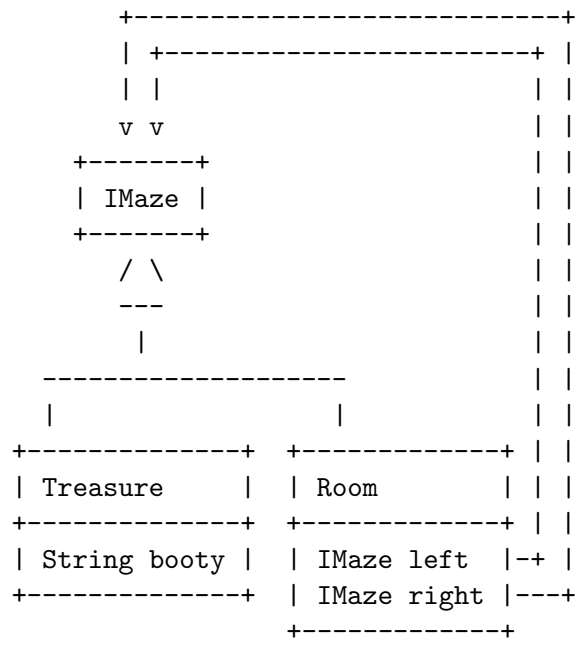
Good luck.

Problem	Points	/
A		/ 0
B		/ 4
C		/ 5
D		/ 6
E		/ 6
F		/ 4
Total		/25

25 Points

Problem 1

Here is a Java class diagram that describes a maze with treasures to hunt for:



A. *(0 points)*

You are not required to do this. You may want to see what the data definitions look like. Do not spend much time on this, unless you do not understand what the data represents. The examples in the next part should make that clear.

Write down the Java class and interface definitions that are represented by this class diagram.

Solution [POINTS 0: 1 point for the interface, 1 point for the class `Treasure`, 1 point for the class `Room`; subtract 1 point if there are no purpose statements]

```
// to represent a maze
interface IMaze{
}

// to represent the booty found in a maze
class Treasure implements IMaze{
  String booty;

  Treasure(String booty){
    this.booty = booty;
  }
}

// to represent a room in a maze
class Room implements IMaze{
  IMaze left;
  IMaze right;

  Room(IMaze left, IMaze right){
    this.left = left;
    this.right = right;
  }
}

// in the class Examples:
  IMaze gold = new Treasure("gold");
  IMaze silver = new Treasure("silver");
  IMaze diamond = new Treasure("diamond");
```

```
IMaze muck = new Treasure("muck");
IMaze mud = new Treasure("mud");

IMaze small = new Room(this.muck, this.silver);
IMaze maze =
  new Room(
      new Room(
          new Room(this.muck, this.silver),
          this.gold),
      new Room(
          this.diamond,
          this.mud));
```

B. *(4 points)*

The following examples represent four different mazes. For the first three mazes we include a description of the maze information that is more readable than the data definition.

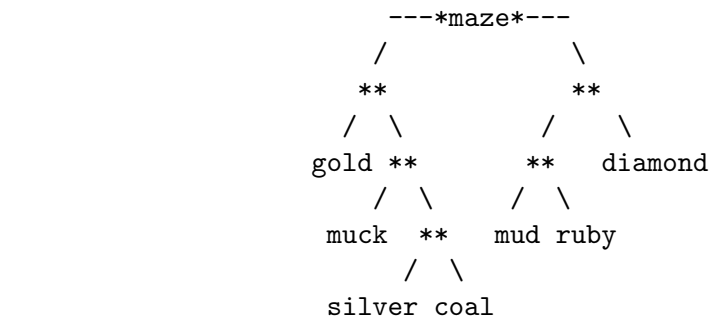
```
// A maze that contains silver:
  IMaze gold = new Treasure("silver");
//-----------------------------------

// A maze that contains muck:
  IMaze muck = new Treasure("muck");
//-----------------------------------

/* small maze:
A maze with one room that contains
muck behind the left door
and silver behind the right door:

           small
          /     \
        muck   silver
*/
  IMaze small = new Room(this.muck, this.silver);
//-----------------------------------------------
```

```
/* a maze:
A maze with one room
leading on the left to a room
  where the left door leads to a room
    where the left door leads to muck
    and right door leads to silver
  and the right door leads to gold
and on the right to a room
  where the left door leads to mud
  and the right door leads to diamond

               *maze*
              /      \
             **       **
            /  \     /  \
          **   gold mud diamond
         /  \
       muck silver
*/
  IMaze maze =
    new Room(
        new Room(
            new Room(this.muck, this.silver),
            new Treasure("gold")),
        new Room(
            new Treasure("mud"),
            new Treasure("diamond")));
//-----------------------------------------

/* a big maze:
*/
  IMaze bigMaze =
    new Room(
        new Room(
            new Treasure("gold"),
            new Room(this.muck,
                     new Room(this.silver,
                              new Treasure("coal")))),
        new Room(
            new Room(new Treasure("mud"),
                     new Treasure("ruby")),
            new Treasure("diamond")));
//-------------------------------------------------------
```

- Describe the big maze (`bigMaze`) in a similar way, either in words, or as a diagram.

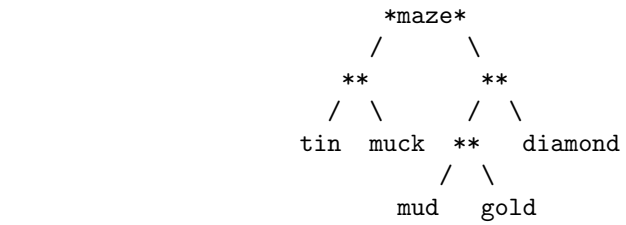
  Solution [POINTS 2: one point if reasonably close, but not correct.]

```
A maze with one room
leading on the left to a room
  where the left door leads to gold
  and right door leads to a room
    where the left door leads to muck
    and the right door leads to a room
      where the left door leads to silver
      and the right door leads to coal
and on the right to a room
  where the left door leads to a room
    where the left door leads to mud
    and the right door leads to ruby
  and the right door leads to diamond
```



- Add an example that represents the following new maze (`newMaze`):

```
A maze with one room
leading on the left to a room
  where the left door leads to tin
  and the right door leads to muck
and on the right to a room
  where the left door leads to a room
    where the left door leads to mud
    and right door leads to gold,
  and the right door leads to diamond
```



Solution [POINTS 2: one point if reasonably close, but not correct.]

```
IMaze newMaze =
  new Room(
      new Room(
          new Treasure("tin"),
          this.muck),
      new Room(
          new Room(new Treasure("mud"),
                   new Treasure("gold")),
          new Treasure("diamond")));
```

C. *(5 points)* Design the method `contains` that determines whether a maze includes the treasure described by a given `String`.

Solution [POINTS 5: 1 point purpose/header; 1 point body in each class; 1 points examples – should include result *true* and result $false$]

```
// in the interface IMaze:
  // does this maze contain the given treasure?
  boolean contains(String treasure);

// in the class Treasure:
  // does this maze contain the given treasure?
  boolean contains(String treasure){
    return this.booty.equals(treasure);
  }

// in the class Room:
  //does this maze contain the given treasure?
  boolean contains(String treasure){
    return this.left.contains(treasure) ||
           this.right.contains(treasure);
  }
// in the class Examples:
  //test the method contains for the IMaze class hierarchy
  boolean testContains(Tester t){
    return
    t.checkExpect(this.gold.contains("gold"), true) &&
    t.checkExpect(this.gold.contains("mud"), false) &&
    t.checkExpect(this.small.contains("muck"), true) &&
    t.checkExpect(this.small.contains("diamond"), false) &&
    t.checkExpect(this.maze.contains("diamond"), true) &&
    t.checkExpect(this.maze.contains("copper"), false);
  }
```

D. *(6 points)*

Design the method `longestPath` that computes the longest path in a maze. The path counts the number of rooms we have to go through to reach the treasure.

Solution [POINTS 6: 1 point purpose/header; 1 point body for the `Treasure` class, 1 point body for the `Room` class, 2 points for definition of the helper method max(purpose + header, body; examples/tests), 1 point for examples for the `longestPath` method.]

```
// in the interface IMaze:
  // compute the length of the longest path in this maze
  int longestPath();

// in the class Treasure:
  // compute the length of the longest path in this maze
  int longestPath(){
    return 0;
  }

// in the class Room:
  // compute the length of the longest path in this maze
  int longestPath(){
    return 1 + this.max(this.left.longestPath(),
                        this.right.longestPath());
  }

  // find the maximum of two given integers
  int max(int a, int b){
    if (a > b){
      return a;}
    else{
      return b;}
  }

// in the class Examples:
  // test the method longestPath for the IMaze class hierarchy
  boolean testLongestPath(Tester t){
    return
    t.checkExpect(this.gold.longestPath(), 0) &&
    t.checkExpect(this.small.longestPath(), 1) &&
    t.checkExpect(this.maze.longestPath(), 3);
  }
```

E. *(6 points)* Design the method `findIt` that helps us find the treasure. It produces a `String` that tells us which way to go to find the given treasure.

For the examples above, we would expect the following answers:

```
this.gold.findIt("silver") ---> ""
this.gold.findIt("gold") ---> "gold"

this.small.findIt("silver") ---> "right silver"
this.small.findIt("gold") ---> ""

this.maze.findIt("silver") ---> "left left right silver"
this.maze.findIt("bronze") ---> ""
```

Notice, that if we cannot find the treasure, the answer is an empty `String`.

You may need the following two facts about the class `String`:

- "abc" + "def" produces a `String` "abcdef"
- the method `s.endWith("abc")` produces `true` if the `String s` ends with the three letters `"abc"` and produces false otherwise.

Solution [POINTS 6: 1 point purpose/header; 1 point body for the `Treasure` class, 2 points body for the `Room` class, 2 points for examples for the `findIt` method.]

```
// in the interface IMaze:
  // produce a description of the path to the given treasure
  String findIt(String treasure);

// in the class Treasure:
  // produce a description of the path to the given treasure
  String findIt(String treasure){
    if (this.booty.equals(treasure)){
      return treasure;
    }
    else{
      return "";
    }
  }
```

```
// in the class Room:
  // produce a description of the path to the given treasure
  String findIt(String treasure){
    if (this.left.findIt(treasure).endsWith(treasure)){
      return "left "+ this.left.findIt(treasure);
    }
    else{ if (this.right.findIt(treasure).endsWith(treasure)){
      return "right " + this.right.findIt(treasure);
    }
    else{
      return "";
    }}
  }

// in the class Examples:
  // test the method findIt for the IMaze class hierarchy
  boolean testFindIt(Tester t){
    return
    t.checkExpect(this.gold.findIt("gold"), "gold") &&
    t.checkExpect(this.gold.findIt("mud"), "") &&
    t.checkExpect(this.small.findIt("muck"), "left muck") &&
    t.checkExpect(this.small.findIt("silver"), "right silver") &&
    t.checkExpect(this.maze.findIt("diamond"), "right left diamond") &&
    t.checkExpect(this.maze.findIt("silver"), "left left right silver") &&
    t.checkExpect(this.maze.findIt("copper"), "");
  }
```

... This page is intentionally left blank ...

F. *(4 points)*

Show the templates for all classes in this problem for which you have designed methods.

Solution [POINTS 4: 1 point template for `Treasure`, 3 points template for `Room`: 1 point for fields, 1 point for methods for fields, 1 point for data types]

```
// in the class Treasure
TEMPLATE:
  FIELDS:
  ... this.booty ...                -- String

  METHODS:
  ... this.contains(String) ...     -- boolean
  ... this.longestPath() ...        -- int
  ... this.findIt(String) ...       -- String

  METHODS FOR FIELDS:


// in the class Room
 TEMPLATE:
  FIELDS:
  ... this.left ...                -- IMaze
  ... this.right ...               -- IMaze

  METHODS:
  ... this.contains(String) ...     -- boolean
  ... this.longestPath() ...        -- int
  ... this.findIt(String) ...       -- String

  METHODS FOR FIELDS:
  ... this.left.contains(String) ...   -- boolean
  ... this.left.longestPath() ...      -- int
  ... this.left.findIt(String) ...     -- String

  ... this.right.contains(String) ...   -- boolean
  ... this.right.longestPath() ...      -- int
  ... this.right.findIt(String) ...     -- String
```