THE UNIVERSITY OF BRITISH COLUMBIA
CPSC 210: MIDTERM EXAMINATION
OCT 4, 2012

Name: _____ Student #: _____

Signature: _____ Lab Section:_____ CS Login ID: _____

Time: 1 hour and 30 minutes

Rules Governing Formal Examinations

1. Each candidate must be prepared to produce, upon request, a UBCcard for identification.
2. Candidates are not permitted to ask questions of the invigilators, except in cases of supposed errors or ambiguities in examination questions.
3. No candidate shall be permitted to enter the examination room after the expiration of one-half hour from the scheduled starting time, or to leave during the first half hour of the examination.
4. Candidates suspected of any of the following, or similar, dishonest practices shall be immediately dismissed from the examination and shall be liable to disciplinary action:
    a. having at the place of writing any books, papers or memoranda, calculators, computers, sound or image players/recorders/transmitters (including telephones), or other memory aid devices, other than those authorized by the examiners;
    b. speaking or communicating with other candidates; and
    c. purposely exposing written papers to the view of other candidates or imaging devices. The plea of accident or forgetfulness shall not be received.
5. Candidates must not destroy or mutilate any examination material; must hand in all examination papers; and must not take any examination material from the examination room without permission of the invigilator.
6. Candidates must follow any additional examination rules or directions communicated by the instructor or invigilator.

Question	Mark	Max
1		5
2		12
3		8
4		8
5		21
6		10
Total		**64**

The following materials are authorized for use during this exam:

- the Java 6 or 7 API

You will also need to checkout the following Java projects from the `/lectures` folder in the repository:

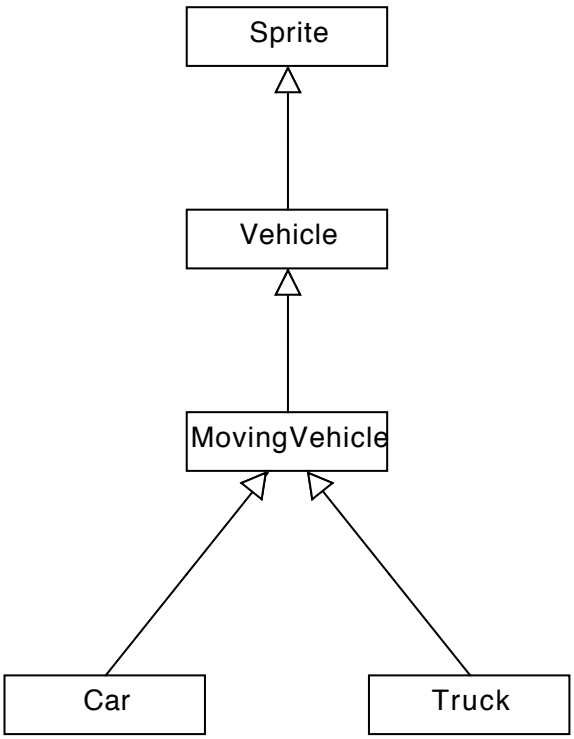
- `JDrawing`
- `SimGame`
- `BouncingVehicles`

No other materials can be consulted.

Note: Question 1 refers to the `BouncingVehicles` project checked out of the repository.

Q1) Type Hierarchies [5 marks]:

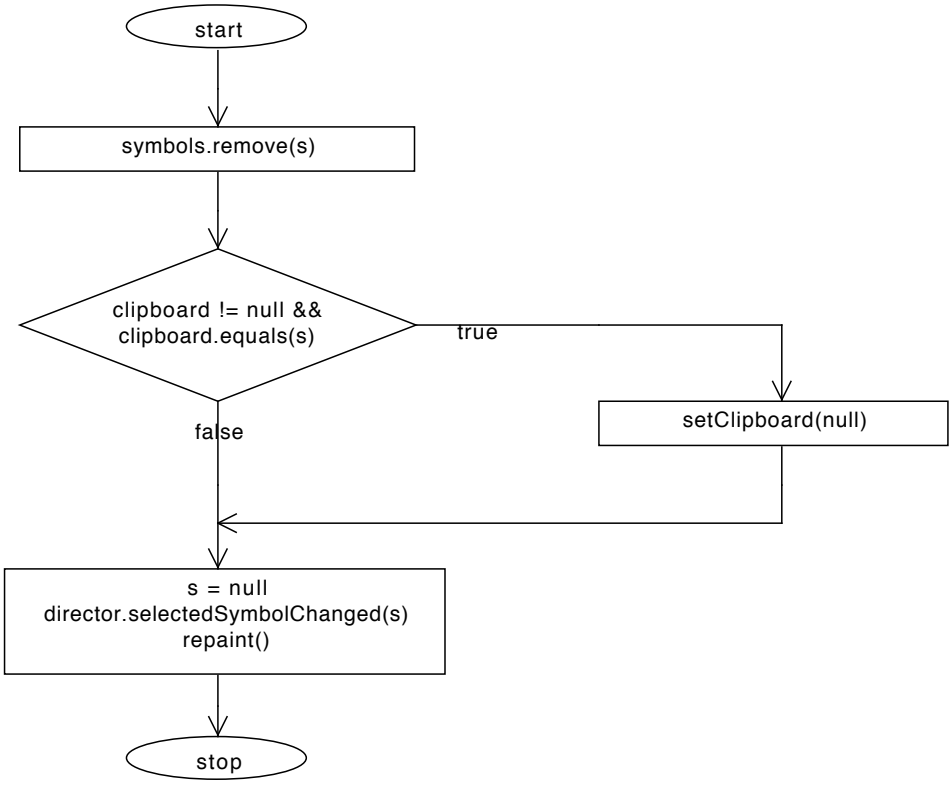
Draw a type hierarchy that includes all of the classes and interfaces in the `ca.ubc.cs210.bouncingvehicles` package. Do not include any class(es) or interface(s) declared in the `ca.ubc.cs210.bouncingvehicles.ui` package or in the Java library.



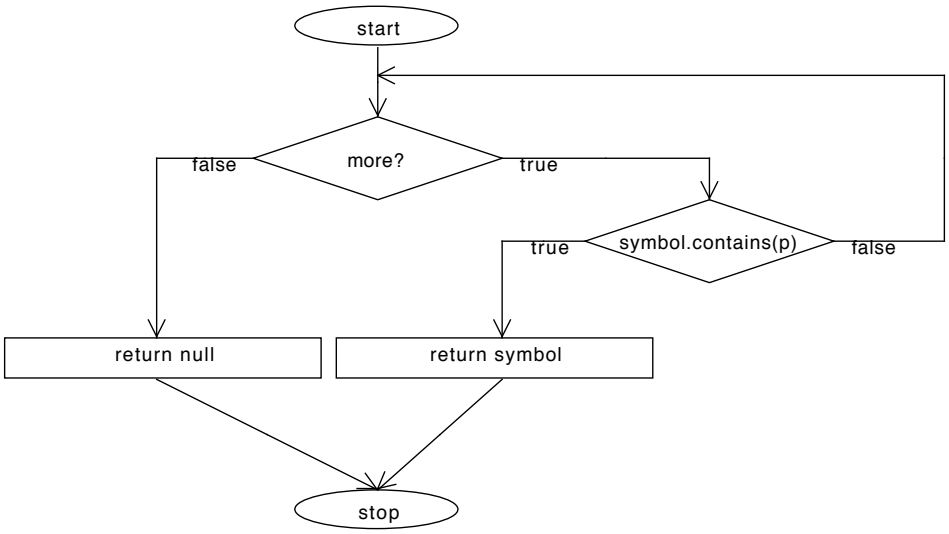
Note: Questions 2 & 3 refer to the `JDrawing` project checked out of the repository.

Q2) Intra-method control flow [12 marks]:
In this question, you may abbreviate statements if you wish, provided you do not introduce any ambiguity into your diagram.

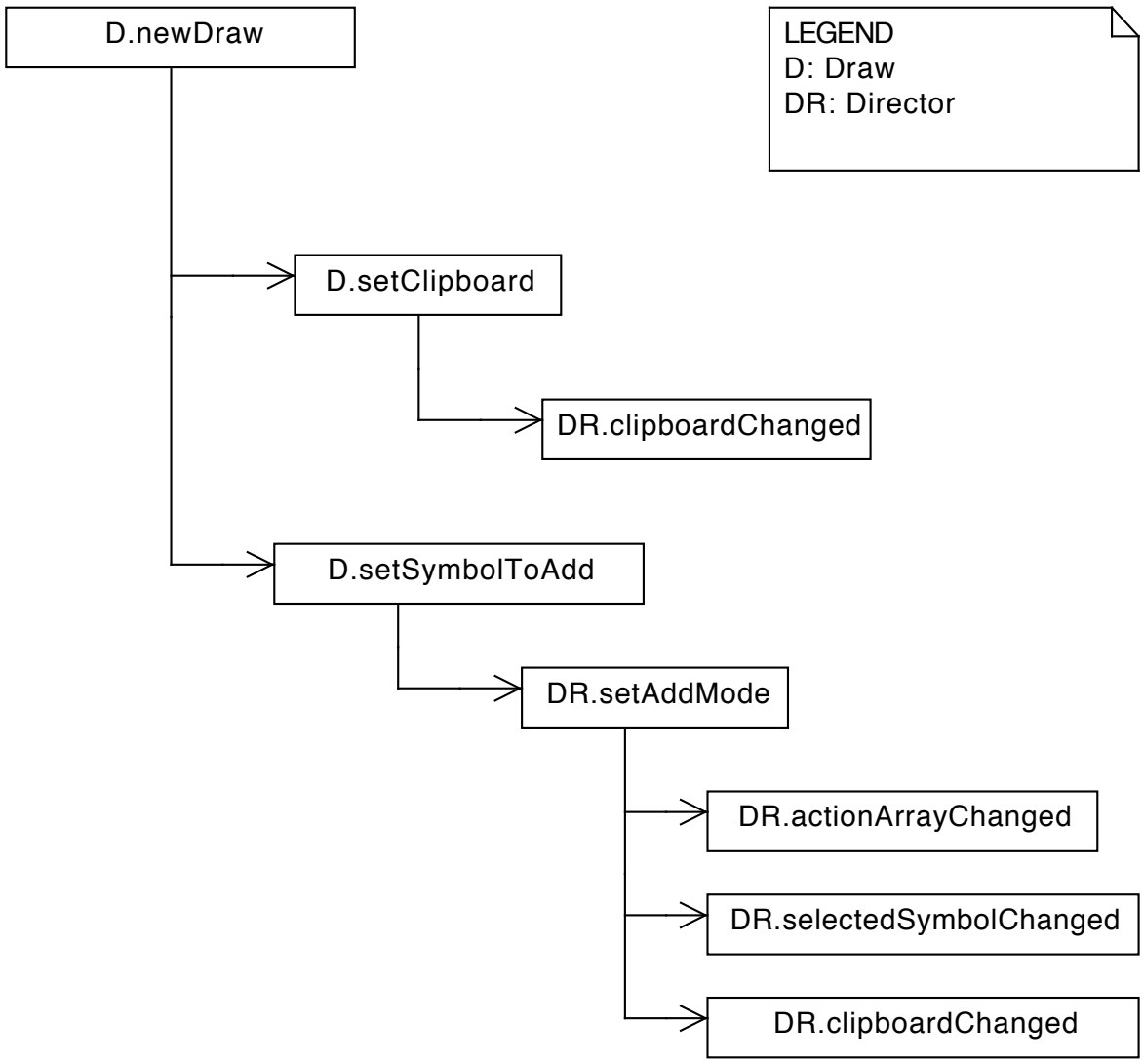
a) [4 marks] Draw an intra-method control flow diagram (flowchart) for `Draw.deleteSymbol`



b) [8 marks] Draw an intra-method control flow diagram (flowchart) for `Draw.getSymbolAt`



Q3) **Inter-method control flow [8 marks]:** Draw a call graph starting at the method `Draw.newDraw`. Do not include methods that belong to classes defined in the Java library. You may wish to rotate the page to accommodate a diagram that is wider than it is tall.



Note: Question 4 refers to the `BouncingVehicles` project checked out of the repository.

Q4) Polymorphism [8 marks]:

Consider the following code:

```
(1) Sprite s;
    MovingVehicle c;
    MovingVehicle t;
    int width;
(2) c = new Car(30.0, 42.0, 45.0, 3.0);
(3) t = new Truck(12.0, 54.0, 135.0, 1.0);
(4) s = t;
(5) width = s.getWidth();
(6) width = c.getWidth();
```

i) [1 mark] What is the actual type of the variable `c` at the statement numbered (2) after the statement executes?

`Car`

ii) [1 mark] What is the apparent type of the variable `c` at the statement numbered (2) after the statement executes?

`MovingVehicle`

iii) [1 mark] What is the apparent type of the variable `s` at the statement numbered (4) after the statement executes?

`Sprite`

iv) [1 mark] What is the actual type of the variable `s` at the statement numbered (4) after the statement executes?

`Truck`

v) [2 marks – one for answer, one for explanation] Is statement (5) legal or illegal? Explain your answer.

Illegal. The `Sprite` interface does not include a method named `getWidth`.

vi) [2 marks – one for answer, one for explanation] Identify the class containing the `getWidth()` method that executes in statement (6). Explain your answer.

`Car`. The actual type of the variable `c` is `Car`, and the `Car` class implements `getWidth()`

Note: Question 5 refers to the `SimGame` project checked out of the repository.

Q5) **Data Abstraction [8 marks]:** The `SimGame` project contains the partial specification and implementation for a `SimPet` class, along with associated unit tests. The `SimPet` represents a pet in a simulated world. Each pet has a location in the two-dimensional world and an energy level. A pet can be pointing in one of only four directions: North, South, East or West. We assume that the pet's location is specified using integer coordinates. In this question, we do not concern ourselves with the size of the world – so we don't worry about pets walking off the edge.

We want to be able to feed the pet and specify the number of units of energy it eats, assumed to be an integer value. We also want to be able to move the pet one unit in whatever direction it is currently pointing. *Each time the pet moves, it consumes one unit of energy*. We also want to be able to rotate the pet left or right by 90 degrees so that it can move in different directions. When a pet rotates, it does not consume any energy. If the pet's energy level drops to zero, it dies.

In this question, you can write your code in Eclipse but you **must** copy it on to this exam paper **before** the end of the exam – there is no electronic submission! Note that it is not necessary to copy the comment statements.

a) **[3 marks]** Write the implementation of the `SimPet` constructor. Run the JUnit tests provided in `ca.ubc.cpsc210.simgame.test.TestSimPet` and ensure that `testConstructor` passes.

```
public SimPet(int x, int y, int initialEnergy) {
     this.x = x;
     this.y = y;
     this.energy = initialEnergy;
     this.direction = 0;
     this.hasHadShots = false;
}
```

b) **[8 marks]** Write the implementation of the `SimPet.move` method. Run the JUnit tests provided and ensure that they all pass.

```
public void move() {
     if (energy > 0) {
          if (direction == 0)
               x = x + 1;
          else if (direction == 1)
               y = y + 1;
          else if (direction == 2)
               x = x - 1;
          else if (direction == 3)
               y = y - 1;
          energy = energy - ENERGY_TO_MOVE;
     }
}
```

c) **[5 marks]** Now suppose we want to add a method that will give a pet its shots. Write the specification for a method `SimPet.giveShots` and include a stub for this method. Assume that a pet can be given its shots only if it has an energy level of at least 5 and hasn't already had its shots. Note that a pet does not consume any energy when it is given its shots. Write your specification in such a way that there is no *requires* clause.

```
// REQUIRES:
// MODIFIES:  this
// EFFECTS:  if this pet is alive and it has not had its shots,
//             then record that the pet has now had its shots
```

d) **[5 marks]** In this part of the question, we ask you to demonstrate how to *use* a data abstraction. Write code that will create a new `SimPet` object located at the origin with 10 units of energy. Your code must then rotate the `SimPet` so that it is pointing west and move it forward 5 steps. Finally, declare a variable of an appropriate type and assign to it the amount of energy that your pet has remaining after rotating and moving.

```
public void doStuff() {
     SimPet p = new SimPet(0, 0, 10);
     p.rotateLeft();          // Or you could
     p.rotateLeft();
     for (int i = 0; i < 5; i++) {
          p.move();
     }
     int remainingEnergy = p.getEnergy();
}
```

Q6) [10 marks] Testing:

Now suppose that you are designing unit tests for the `SimPet.feed` method. Outline the tests that you would conduct. Note that it is not necessary to write formal JUnit tests but you must sketch out the code in sufficient detail that you clearly identify the methods that will be called and the checks (assertions) that you will conduct for each test. Start by specifying the code that you assume will run before each of the tests:

[2 marks] Code that runs before each test:

```
SimPet p = new SimPet(0, 0, 1);
```

[8 marks] The unit tests:

```
// Feed a dead pet
p.move();
p.feed(1);
assertEquals(0, p.getEnergy());

// Feed with "typical" negative energy
p.feed(-5);
assertEquals(1, p.getEnergy());

// Feed with 0 energy - a boundary case
p.feed(0);
assertEquals(1, p.getEnergy());

// Feed with 1 energy - a boundary case
p.feed(1);
assertEquals(2, p.getEnergy());

// Feed with 5 energy - a "typical" case
p.feed(5);
assertEquals(6, p.getEnergy());
```