**CSE 220 Homework Assignment 4 (Due 7/23/24)**

**1) (50 pts)** *(Polymorphism)* This problem deals with polymorphism in Java.   
a. Explain what a **polymorphic reference** is in Java, and why it is useful in practice.

A polymorphic reference in Java is a variable that can refer to multiple types of objects at different points in time. This allows flexibility in the code written, and allows the developer to write more readable and concise code. As we learned in lecture, it might be useful to refer to the same variable in different contexts. For instance, you might pass some variable into a function in a program and want that program to execute the same logic for any of a set of object types that variable might references but use some overridden function depending on the type of object that variable actually does reference.

b. Explain the principal differences between **inheritance-based polymorphism** and **interface-based** **polymorphism** in 2-4 sentences.

Regardless of whether the implementation is interface-based or inheritance-based, polymorphism generally relates to the concept of the many “forms” an object could take within a program.

Inheritance based polymorphism obviously refers to class based inheritance. Flexibility enabled by polymorphism might be accessed by inheritance of the classes that polymorphic reference could instantiate. Through the actual instantiation of the object as one or another class, the object may be able to use the attributes or methods of that class. It may also be able to access the methods or attributes of any ancestral class.

Interface based polymorphism uses an interface to

c. Assume the following classes and interface are defined:  
- Class **Camera**, which has methods **void record()** and **void snapShot()**  
- Interface **isMobile**, which declares methods **void** **moveFront(double)** and   
**void rotate(int)**  
- Class **CamDrone**, which extends *Camera*, implements *isMobile*, and has the additional method **void recharge()**.

Assuming neither Camera nor CamDrone are abstract, all methods have been implemented correctly, and the class constructors for Camera and CamDrone have no input arguments, indicate which of the following snippets of code (i. – vi.) will work correctly, and which will not. **Make sure you explain each answer in 1-2 sentences.**  
**i. Camera ca1 = new Camera();  
 ca1.recharge();  
ii. Camera ca2 = new CamDrone();  
 ca2.snapShot();  
iii. CamDrone cd1 = new Camera();  
 cd1.record();  
iv. CamDrone cd2 = new CamDrone();  
 cd2.rotate(90);  
v. isMobile im1 = new isMobile();  
 im1.moveFront(1.0);  
vi. isMobile im2 = new CamDrone();  
 im2.recharge();  
  
2) (50 pts)** *(JavaFX and Events)* For this problem you are going to be assembling a collection of shapes in a JavaFX scene and adding a simple interactive control.  
a. You are first to draw a 2d car in a JavaFX scene with the following properties:  
- The car needs a body/chassis, which should be constructed using the **Polygon** class.  
- The car needs two wheels (or more, if you want!) which should be **Circle**s or **Ellipse**s.  
- The car needs windows, which can be constructed using any shape(s) of your choice.   
- All of the car components above should have distinct fill colors (chassis should be different color from tires, which should be different in color from the windows.)

You can include other elements in your scene if you like, such as road, sky, etc. A good car should probably also have doors, so feel free to include those if you like – they are not strictly necessary, however.

b. You are to add *at least one* **non-button** **control** to your scene that allows you adjust a property or element of the car in some way. Examples include, but are not limited to, altering the size or color of a car component in response to a slider or color control, adjusting the shape of the car windows with a slider, etc. You must employ *at least* one of the following in your code to address this: i. An event-handler, ii. Property-binding, or a iii. A ChangeListener. Note that option i. is the easiest to implement, but options ii and iii may be of interest to those trying to use advanced GUI automation.

**Responses to Problems #1 should be in .doc(x) or .pdf format. Upload a .zip file containing this file and your .java file(s) for Problem #2 to Blackboard, using the filename “LN\_FN\_4.zip” where LN is your last name and FN is your first name.**