Topics: Polymorphism, Interfaces
Ref: CSC20 PAL WS Inheritance

Polymorphism

We worked through the example of the base class CircularSolid and derived classes Cylinder and Cone in the previous worksheet on inheritance. The driver that we created instantiated single objects of Cylinder or Cone type. We used the following kind of declaration:

```
Cylinder cl = new Cylinder()
Cone co= new Cone();
```

It turns out that since Cylinder *"is a"* CircularSolid and Cone *"is a"* CircularSolid we can also do the following kind of declaration:

```
CircularSolid cl = new Cylinder()
CricularSolid co= new Cone();
```

So, the variable *cl* and *co* are of type CircularSolid, but point to an object in memory that is a Cylinder or a Cone. This is possible because the objects of a derived class are the same size or larger than an object of the parent class.

With this, if we needed to store and handle multiple 'CircularSolid' objects in one place, we can declare an array of type CircularSolid, which stores types of both child classes as follows:

```
CircularSolid[] circSolids = new CircularSolid[]{

new Cylinder(2,3), new Cylinder(5,2), new Cone(1,4)};
```

We can now call class methods like *'calculateVolume()'* on any object retrieved from the array:

```
for (int i = 0; i<3; i++){
     volume = circSolids[i].calculateVolume();
     System.out.println("This object's volume: "+ volume);}
```

The beauty of it is that not only can we neatly store different child objects (of a common parent class) in a single array, but we are also easily calling the same common method (how each class object specifically implements it) on different objects of the array and have each object exhibit its own unique implementation or behavior at runtime. This is called ***polymorphism***. It uses a feature called dynamic binding or late binding which calls the appropriate instance method depending on which object is being used at runtime.

Ex. 1) Implement in Java a fruit-picking program, where 'Fruit' is the parent class, and 'Apple, 'Banana, and 'Plum' are child classes. 'Fruit' should have a variable to count how many you've picked.

The child classes should each implement method 'computeCost()', which is unique to each fruit. Apples always cost twice as many as you've picked in dollars. Bananas are free unless you've picked more than ten, in which case they're fifteen dollars total. Plums are always three dollars each.

Create a 'FruitBasket' class which provides an array for Fruit objects (apple/banana/plum), capable of storing all three fruits together. The FruitBasket class should also have a method called printCost that inputs a Fruit Array as follows:

double printCost(Fruits [] fruitArray)

that print the total cost of each type of fruit, by calling 'computeCost()' on each element of the array. It also returns the grand total for the entire fruit basket (array).

Ex. 2 What do you think are the benefits of polymorphism? Give an example using another class relationship. Create a polymorphic method in your driver that has a single object parameter (not an array) declared to be of parent type but at run time is able to permit an object of any of its child classes.

Use the **instance of** function in this method to determine the actual type of the object passed.

Ex. 3 Create a method in the child class that is unique to it and does not occur in the parent class. Try calling this method in the polymorphic method.

Ex. 4 Read up on how to cast objects. Can you cast a parent object into a child object or the other way around? How does it work? Use casting to see if you can solve the problem in Ex 3.

Interfaces: Due to the complexities inherent in multiple inheritances, Java does not offer this feature. However, it permits use of interfaces. An interface in Java is a blueprint of a behavior and is used to implement abstraction, so it contains abstract methods ie. method signatures only. This means it can specify what a class must do but not how. More than one interface can be "implemented" by another class thereby allowing this class to exhibit behaviors of all these interfaces that it has chosen to implement. In fact, this in class need to define all methods of all of the interfaces that it implements. An Interface do not have constructors so they cannot be instantiated as objects. Here is an example of an interface and its use:

```
 interface Vehicle {

   // all are the abstract methods.
   void accelerate(int a);
   void brake(int a);}

class Car implements Vehicle{

int speed;

public void accelerate(int incSpeed){ // to speed up
    speed = speed + incSpeed;
}
public void brake(int decSpeed){ // to slow down
     speed = speed - incSpeed;
}
```

.Ex. 5 Write an interface called Pet that exhibits the behavior of a pet. The pet can eat, play, sleep. Write two classes Cat and Dog that implement Pet Write a driver that creates some dog and cat objects.

Ex. 6. Update the Cat and Dog class with a compare method that can be used to compare objects. You will need to implement the Comparator interface to do this. E.g. class Dog implements Comparator<Dog> Read about this in your textbook. Use this to compare dogs by age in your driver.