COE318 Lecture Notes Week 9 (Week of Oct 27, 2014)

Zoo Version 3: Abstract classes and methods

- `AbstractAnimal` is just a normal class and it is legal to write:

  ```
  AbstractAnimal a = new AbstractAnimal("Sue");
  ```

- This should not be allowed!
- It makes no sense to create an animal when we don't know what kind of animal it is.
- `AbstractAnimal` exists only for the purpose of subclassing.
- We can make this clear by declaring it to be `abstract`.
- The compiler will now disallow:

  ```
  AbstractAnimal a = new AbstractAnimal("Sue");
  ```

- We now refer to `AbstractAnimal` as an *abstract class* that cannot be instantiated.
- But `Lion` and `Hedgehog` are *concrete* classes that can be instantiated.
- We can also declare methods to be abstract. If we do so, we **must** declare the class to be abstract.
- Furthermore, when we extend an abstract class as a concrete class, the compiler will insist that we write implementation code for any abstract methods.
- The code for our classes is given below.
  - Note: the Zoo class needs no changes.
  - The AbstractAnimal class is now declared "abstract" as is its "noise()" method.
  - Both Lion and Hedgehog now have to implement (and not just inherit) the "noise()" method.

```
public abstract class AbstractAnimal {
   private final String name;
   private final int birthYear;

   public AbstractAnimal(String name, int year) {
      this.name = name;
      birthYear = year;
   }

   public int age() {
      return 2014 - birthYear;
```

```
      }

      public String getName() {
         return name;
      }

      public abstract String noise();

      public boolean eatsPeople() {
         return true;
      }

   }
```

```
   public class Hedgehog extends AbstractAnimal {

      public Hedgehog(String name, int year) {
         super(name, year);
      }

      @Override
      public boolean eatsPeople() {
         return false;
      }

      @Override
      public String noise() {
        return "Chirp...chirp";
      }
   }
```

```
   public class Lion extends AbstractAnimal {

      public Lion(String name, int year) {
         super(name, year);
      }

      @Override
      public String noise() {
         return "Rrroar (yum yum)";
      }
   }
```

Zoo Version 4: Interfaces

- The zoo owner is happy with version 3; lots and lots of concrete animal classes can be easily added; no changes needed to the Zoo or AbstractAnimal classes.
- But then one day he says, “I love my fridge. I want to have it as one of the amimals in my zoo.”
- The guy may be a lunatic, but we have to respect his wishes.
- “Tell him just to subclass Refrigerator from AbstractAnimal.”
- But he can't do that; Refriger ator is already subclassed from ElectricalAppliance.
- OK, we'll make an *interface* called `Animal`. He just has to *implement* the interface.

Casting (reference data types) and `instanceof` operator

```
/*
 * Copyright (c) 1995, 2008, Oracle and/or its affiliates.
All rights reserved.
 *
 * Redistribution and use in source and binary forms, with
or without
 * modification, are permitted provided that the following
conditions
 * are met:
 *
 *   - Redistributions of source code must retain the above
copyright
 *     notice, this list of conditions and the following
disclaimer.
 *
 *   - Redistributions in binary form must reproduce the
above copyright
 *     notice, this list of conditions and the following
disclaimer in the
 *     documentation and/or other materials provided with
the distribution.
 *
 *   - Neither the name of Oracle or the names of its
 *     contributors may be used to endorse or promote
```

```
products derived
 *     from this software without specific prior written
permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS "AS
 * IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO,
 * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE COPYRIGHT
OWNER OR
 * CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR
 * PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF
 * LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR
TORT (INCLUDING
 * NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE
USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
DAMAGE.
 */

class InstanceofDemo {
    public static void main(String[] args) {

        Parent obj1 = new Parent();
        Parent obj2 = new Child();

        System.out.println("obj1 instanceof Parent: "
            + (obj1 instanceof Parent));
        System.out.println("obj1 instanceof Child: "
            + (obj1 instanceof Child));
        System.out.println("obj1 instanceof MyInterface: "
            + (obj1 instanceof MyInterface));
        System.out.println("obj2 instanceof Parent: "
            + (obj2 instanceof Parent));
        System.out.println("obj2 instanceof Child: "
            + (obj2 instanceof Child));
        System.out.println("obj2 instanceof MyInterface: "
+ (obj2 instanceof MyInterface));
```

```
        System.out.println("obj2 instanceof Object: "
            + (obj2 instanceof Object));
        System.out.println("obj1 instanceof Object: "
            + (obj1 instanceof Object));
    }
}

class Parent {}
class Child extends Parent implements MyInterface {}
interface MyInterface {}
```

What is an interface?

- An interface is simply a collection of zero or more public abstract methods.
- A class can implement as many interfaces as it wants. It just has to ensure that all the abstract methods are actually implemented. (And the compiler will ensure that this is done.)
- We also take advantage of the Animal interface and have AbstractAnimal implement it.
- Objects can be declared as their own type or as any of their superclass's type.
- In addition, an object can be declared as any interface type implemented by the actual class.
- Consequently, we can now declare all the animals (and the fridge) as type `Animal`.
- We also have AbstractAnimal implement a standard imterface, Comparable<Animal>.

```
    interface Animal {
       String noise();
       boolean eatsPeople();
       int age();
       String getName();
    }
    public abstract class AbstractAnimal
            implements Animal,
            Comparable<Animal> {

       private final String name;
       private final int birthYear;

       public AbstractAnimal(String name, int year) {
          this.name = name;
          birthYear = year;
       }

       @Override
       public int age() {
          return 2014 - birthYear;
       }

       @Override
       public String getName() {
          return name;
       }

       @Override
       public abstract String noise();

       @Override
       public boolean eatsPeople() {
          return this instanceof Dangerous;
       }

       @Override
       public int compareTo(Animal other) {
          return getName().compareTo(other.getName());
       }

       @Override
       public boolean equals(Object obj) {
          if (obj == null) {
             return false;
          }
          if (getClass() != obj.getClass()) {
             return false;
          }
          final AbstractAnimal other =
                  (AbstractAnimal) obj;
          if ((this.name == null)
                  ? (other.name != null)
                  : !this.name.equals(other.name)) {
             return false;
          }
          return true;
       }

       @Override
       public int hashCode() {
          int hash = 7;
          hash = 83 * hash
                  + (this.name != null
                  ? this.name.hashCode() : 0);
          return hash;
       }
    }
```

```
    public interface Dangerous {

    }
```

```
    public class Lion extends AbstractAnimal
       implements Dangerous{

       public Lion(String name, int year) {
          super(name, year);
       }

       @Override
       public String noise() {
          return "GGGrowl";
       }

    }
```

```
public class Hedgehog extends AbstractAnimal {

   public Hedgehog(String name, int year) {
      super(name, year);
   }

   @Override
   public String noise() {
     return "Chirp chirp";
   }
}
```

```
import java.util.ArrayList;

public class Zoo {

   private final ArrayList<Animal> animals =
           new ArrayList<Animal>();

   public void add(Animal animal) {
      animals.add(animal);
   }

   public String getNames() {
      String s = "";
      for (Animal a : animals) {
         s += a.getName() + "\n";
      }
      return s;
   }

   public Animal[] getAnimals() {
      Animal[] ts = new AbstractAnimal[0];
      return animals.toArray(ts);
   }

   public static void main(String[] args) {
      Zoo zoo = new Zoo();
      Animal a = new Lion("Alex", 2010);
      zoo.add(a);
      zoo.add(new Lion("Betty", 2009));
      zoo.add(new Hedgehog("Charlie", 2010));
      System.out.println(zoo.getNames());
```

```
          Animal[] animals = zoo.getAnimals();
          for (Animal an : animals) {
             System.out.println(an.getClass().getName()
                     + " " + an.getName() + ": "
                     + (an.eatsPeople()
                     ? "eats" : "does not eat")
                     + " people");
          }
       }
    }
```

Zoo Version 5: Exceptions

```
    import java.util.ArrayList;

    public class Zoo {

       private final ArrayList<Animal> animals =
               new ArrayList<Animal>();

       public void add(Animal animal) {
          if (animals.contains(animal)) {
             throw new IllegalArgumentException("Duplicate
name not allowed");
          }
          animals.add(animal);
       }

       public String getNames() {
          String s = "";
          for (Animal a : animals) {
             s += a.getName() + "\n";
          }
          return s;
       }
```

```
        public Animal[] getAnimals() {
           Animal[] ts = new AbstractAnimal[0];
           return animals.toArray(ts);
        }

        public static void main(String[] args) {
           Zoo zoo = new Zoo();
           Animal a = new Lion("Alex", 2010);
           zoo.add(a);
           zoo.add(new Lion("Betty", 2009));
           Animal charlie1 = new Hedgehog("Charlie", 2010);
           zoo.add(charlie1);
           System.out.println(zoo.getNames());
           Animal charlie2 = new Lion("Charlie", 2007);

           try {
              zoo.add(charlie2);
           } catch (IllegalArgumentException ex) {
              System.err.println("Duplicate name not
added");
           }

           Animal[] animals = zoo.getAnimals();
           for (Animal an : animals) {
              System.out.println(an.getClass().getName()
                      + " " + an.getName() + ": "
                      + (an.eatsPeople()
                      ? "eats" : "does not eat")
                      + " people");
           }
        }
     }
```