Day 13: Abstract Classes

1 more challenge to get your next star!

Points: 14/15

30 Days of Code

Problem Submissions Leaderboard Editorial Tutorial

Day 13 of Code: Abstract Classes + Casting!

Terms you'll find helpful in completing today's challenge are outlined below, along with sample Java code (where appropriate).

Abstraction

This is an essential feature of object-oriented programming. In essense, it's the separation between what a class does and how it's accomplished.

One real world example of this concept is a snack machine, where you give the machine money, make a selection, and the machine dispenses the snack. The only thing that matters is what the machine does (i.e.: dispenses the selected snack); you can easily buy a snack from any number of snack machines without knowing how the machine's internals are designed (i.e.: the implementation details).

Abstract Class

This type of class can have abstract methods as well as defined methods, but it cannot be instantiated (meaning you cannot create a new instance of it). To use an abstract class, you must create and instantiate a subclass that extends the abstract class. Any abstract methods declared in an abstract class must be implemented by its subclasses (unless the subclass is also abstract).

The Java code below demonstrates an abstract Canine class and 2 of its canine breed subclasses, KleeKai and SiberianHusky:

```
/** Superclass **/
abstract class Canine{
    // instance variables
    String name;
    String color;
    String gender;
    int age;

    /** Parameterized Constructor
 *  @param name Dog's name
 *  @param color Dog's color
 *  @param age Dog's age
 *  @param mF Dog's gender ('M' for male, 'F' for female)
 **/
    Canine(String name, String color, int age, char mF){
        this.name = name;
        this.color = color;
        this.age = age;
        this.gender = (mF == 'M') ? "Male " : "Female ";
```

```
        }

        /** Abstract method declaration
     *  @return Implementations should return a string describing the breed **/
        abstract String getBreed();

        /** Defined method **/
        void printInfo(){
            // print information about the dog:
            System.out.println(name + " is " + ((age%10 == 8)? "an " : "a ") + age + " year old "
                + gender + getBreed() + " with a " + color + " coat.");
            // note: the '(age%10 == 8)' conditional ensures grammatical correctness if dog is 8 or 18; dogs do not live longer than this.
        }
    }

    //** Subclass of Canine **/
    class KleeKai extends Canine{
            /** Parameterized Constuctor **/
        KleeKai(String name, String color, int age, char mF){
            super(name, color, age, mF);
        }

        /** Abstract method implementation
     *  @return "Klee Kai" **/
        String getBreed(){ // abstract method implementation
            return "Klee Kai";
        }
    }

    /** Subclass of Canine **/
    class SiberianHusky extends Canine{
            /** Parameterized Constuctor **/
        SiberianHusky(String name, String color, int age, char mF){ // Constructor
            super(name, color, age, mF);
        }

        /** Abstract method implementation
     *  @return "Siberian Husky" **/
        String getBreed(){ // abstract method implementation
            return "Siberian Husky";
        }
    }
```

The Canine class has `1` abstract method, abstract void getBreed(), and `1` defined method, void printInfo(). Because an abstract class is not fully defined, attempting to instantiate it like so:

```
Canine myPuppy = new Canine("Lilah", "Grey/White", 5, 'F');
```

results in `error: Canine is abstract; cannot be instantiated`. This type of class is only meant to serve as a base or blueprint for connecting the subclasses that inherit (extend) it. While we can't instantiate Canine, we can instantiate its subclasses, KleeKai and SiberianHusky. This code:

```
Canine c = new KleeKai("Lilah", "Grey/White", 5, 'F');
Canine d = new SiberianHusky("Alaska", "Grey/Black/White", 16, 'F');
c.printInfo();
d.printInfo();
```

executes and produces this output:

```
Lilah is a 5 year old Female Klee Kai with a Grey/White coat.
Alaska is a 16 year old Female Siberian Husky with a Grey/Black/White coat.
```

because c and d are polymorphic references objects of Canine's subclasses.

Additional Language Resources

C++ Abstract Base Classes