July 6, 2010

Allegory of the Cave

- Related: The_Treachery_of_Images
- Related: Allegory_of_the_cave

Plato enjoyed telling the story of a cave with prisoners (since the day they were born) who where chained to their seats and facing a wall. Behind them was a bonfire. Between the prisoners and the bonfire, free people would walk past and hold up objects so that the shadow of the object appeared on the wall. At no point could someone turn around an look at what was behind them. The free people would speak, and the sound would echo off the cave wall and the prisoners believed that the shadows were making these sounds. The prisoners, who could only speak, would play a game to guess the name of the image on the cave wall. Those who did well were more popular. Those who did not name objects well were less popular.

Theory of Forms

Related: Theory_of_forms

Plato believed everything has a form, and that form is a blueprint of the perfection. The that the material world as it seems to us is not the real world, but only a shadow of the real world.

1. Everything in creation is known as a particular of that form. A particular cannot exist without a form. The form must exist prior to the particular being made. (Modularity)
2. Form does not exist in this world. We have no way of seeing the full definition of any form. They have no time, for they are eternal. They have no space, for they are not physical. (Abstraction)
3. We are only able to make clear the details of a form by studying the form. We do not the know true form of any particular. (Encapsulation)

Object-Oriented Design Principles

In computer science, we often write things which are called "libraries." A library is a collection of tools which perform various task. If you want to do something in a program, you should research if your desired task has already been made. If it has, use that code instead. Do not try to reinvent the wheel unless you can improve upon the wheel. Do not write what has already been written.

In Java, we typically use classes to denote a library. A class is a composed of fields and methods. Fields are properties to a class. They retain information about an instance of a class. Methods of a class can only be applied to an instance of that class, and can only affect fields of an instance of a class.

In designing a class, we aim for three principles: Modularity, Abstraction, and Encapsulation.

- Modularity: The different components of the class are organized into different methods. Classes should have their own fields so that two objects do not share the same memory space. This modular approach keeps things tidy and allows users to make multiple versions of an object without concern for how different objects might interact.
- Abstraction: The methods and fields should represent a generic version of an object. An instance of an object should then be modified to be more specific, but still work with all of the methods in the designed class.
- Encapsulation: At no point do you describe how the internal methods or fields work. The end user doesn't care how it works, as long as it works. By not telling the end user how things are done, the class can be easily changed without the end user being aware that a change was made.

In Plato's Theory of Forms, he proposed three ideas for how things are planned. In object oriented programming, we use the same three ideas for planning objects. Plato wanted to understand how objects are developed, and we want to understand how to develop objects. Both of us are looking at the same coin from different sides.

Object-Oriented Design Goals

Our software should achieve the following:

- Robustness: Software should be able to handle correct input as well as input that is correct but might not be in the form we intended. A good example of this is how we write dates:
    - July 2, 2007
    - Jul 2, 2007
    - 07/02/2007
    - 07/02/07
    - 2 July 2007
    - 07-02-2007
    - 07022007
    - 2007-07-02
    - 20070702
- Adaptability: Our library should be easily changed over time to adapt to new technologies that we cannot conceive at this very moment. On the Internet, new technologies are being produced daily, and if the software on your machine is not adaptable, you will not be able to experience this new technology.
- Reusability: Ideally, our code is not single purpose. We should be able to take our library and apply it to many problems. Other people should be able to take our library and apply it to problems that we did not anticipate. The real definition of a "tool" is anything that the user creates a new use for that the maker did not intend.

Designing a class

A class is composed of three things: a name, a collection of fields, and a collection of methods. The methods are allowed to work with any fields of a class and input parameters. A class should represent some NOUN. The methods in a class should represent some verb.

```
class Bicycle {
    int gear;
    int speed;

    Bicycle() {
        gear = 1;
        speed = 0;
    }

    int getSpeed() {
        return speed;
    }

    int getGear() {
        return gear;
    }

    void setSpeed(int speed) {
        this.speed = speed;
    }

    void setGear(int gear) {
        this.gear = gear;
    }
```

```
        public String toString() {
                return "Gear: "+gear+" Speed: "+speed;
        }
}
```

In testing this library, we can do the following.

```
class bikeTest {
        public static void main(String[] args) {
                Bicycle b = new Bicycle();
                System.out.println(b);

                b.setSpeed(4);
                System.out.println(b);

                b.setGear(2);
                System.out.println(b);
        }
}
```

Static versus Non Static Context

The goal of this course is to get you to be comfortable with object oriented programming.

A class is the definition of a data structure. An object is the instantiation of a class.

```
class My_Class  {

        static int my_class_field;
        int my_object_field;

        My_Class() {
                my_class_field = 10;
                my_object_field = 10;
        }

      void reset() {
                my_class_field = 10;
                my_object_field = 10;
      }

        static void my_class_method () {
                my_class_field = 20;
        }

        void my_object_method () {
                my_object_field = 20;
            }

            void effect_both () {
                my_class_field = 30;
                my_object_field = 30;
            }

            void printFields() {
                        System.out.println("my_class_field  =  "+my_class_field+",  my_object_field  =
"+my_object_field);
            }
      }
      public class static_test {
            public static void main(String[] args) {

                My_Class a = new My_Class();
                My_Class b = new My_Class();

                System.out.println("");
                System.out.println("First pass - Only a's non-static field is effected");
                a.my_object_method();
                a.printFields();
                b.printFields();

                a.reset();
                b.reset();

                System.out.println("");
                System.out.println("Second pass - Both a's and b's static fields are effected");
                a.my_class_method();
                a.printFields();
                b.printFields();

                a.reset();
                b.reset();

                System.out.println("");
                 System.out.println("Third pass - Both a's and b's static fields are effected and a's non-
static field is effected");
                a.effect_both();
                a.printFields();

                b.printFields();

          } // End main

     }
```

What does this look like if we run it:

```
First pass - Only a's non-static field is effected
my_class_field = 10, my_object_field = 20
my_class_field = 10, my_object_field = 10

Second pass - Both a's and b's static fields are effected
my_class_field = 20, my_object_field = 10
my_class_field = 20, my_object_field = 10

Third pass - Both a's and b's static fields are effected and a's non-static field is effected
my_class_field = 30, my_object_field = 30
my_class_field = 30, my_object_field = 10
```