11/4/22

5.1
Anatomy of a Java Class

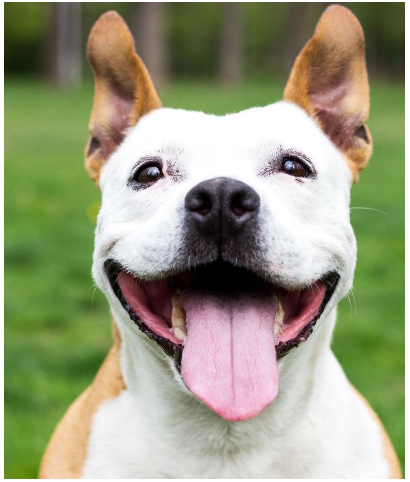
Definitions

Class - Blueprint for an object; instructions for how construct an object. **There can be ONLY ONE of these -> "Dog"**

Object - A particular instance of a class; use the new operator to create an object instance from a Class. **There can be MANY of these -> "A 3 year-old German Shepherd named Roscoe", "A 1 year-old Golden Retriever named Lucy"**

Properties and Methods - **Properties** are attributes (name, age, breed) and **Methods** are operations (play, eat, sleep); and each can be either `public` (available outside the Object) or `private` (available from only the inside of an Object)

Example Class



```
// Defining a Class
public class Dog {
    public Dog(String name, int age) {
    }
}


// Creating Objects
Dog scout = new Dog("Scout", 10);
Dog bailey = new Dog("Bailey", 5);
```

Writing a Class

```
public class Dog {
    // Attributes
    public String name;
    private int age;

    // Constructor
    public Dog(String name, int age) { … }

    // Methods
    public int getAge() { … }
    public void feedDog() { … }
    private int calcFoodAmount() { … }
}
```

Instance Variables

- Also known as attributes, properties, or fields
- Holds the data of an object
- **Every Object instance has their own values for these properties**

```
Dog scout = new Dog("Scout", 10);
Dog bailey = new Dog("Bailey", 5);


scout.name.equals("Scout") == true
bailey.name.equals("Bailey") == true

scout.name.equals(bailey.name) == false
```

Instance Methods

- Define the behavior generically in the Class
- **So that it can be used by every Object instance**

```
public void feedDog() {
    System.out.println("Gave " + name + " a bowl of food.");
}
```

```
Dog scout = new Dog("Scout", 10);
Dog bailey = new Dog("Bailey", 5);

scout.feedDog();
"Gave Scout a bowl of food."

bailey.feedDog();
"Gave Bailey a bowl of food."
```

Private vs Public

Private

An instance variable or method that can only be accessed within the class

- On the AP Exam all instance variables should be private
- Some methods can be private if they are only used internally

Public

An instance variable or method that can be accessed outside of a class like in the main method

- Most methods are public

```java
public class Dog {
    // Attributes
    public String name;
    private int age;

    // Constructor
    public Dog(String name, int age) { … }

    // Methods
    public int getAge() { … }
    public void feedDog() { … }
    private int calcFoodAmount() { … }
}
```

***public** properties and methods can be directly accessed from outside an object*

OKAY

```java
Dog scout = new Dog("Scout", 10);
scout.name = "S-Dog";
scout.getAge();
scout.feedDog();
```

ERROR

```java
scout.age = 5;
scout.calcFoodAmount();
```

```
public class Dog {
    // Attributes
    public String name;
    private int age;

    // Constructor
    public Dog(String name, int age) { … }

    // Methods
    public int getAge() { … }
    public void feedDog() { … }
    private int calcFoodAmount() { … }
}
```

***private** properties and methods can be accessed only from within an object*

***Question:** Why might we want code outside Dog from modifying age or calling calcFoodAmount()?*

Object-Oriented Design

A design philosophy used by programmers when developing larger programs

1. Decide what classes you’ll need to solve a problem
2. Define the data (instance variables) and functionality (methods) for the classes
3. Utilize classes and objects to solve your problem

Data Encapsulation

Data (instance variables) and the code acting on it (methods) are wrapped together in a single implementation and the details are hidden.

Data is safe from harm by keeping it private

```java
public class Dog {
  // Attributes
  public String name;
  private int age;
  // Constructor
  public Dog(String name, int age) {
    this.name = name;
    this.age = age;
  }
  // Methods
  public int getAge() {
    return age;
  }
  public void feedDog() {
    System.out.println("Gave " + name + " " + calcFoodAmount() + " of food.");
  }
  private String calcFoodAmount() {
    if (age < 6) {
      return "3 bowls";
    }
    return "1 bowl";
  }
}
```

```
public class Dog {
  // Attributes
  public String name;
  private int age;
  // Constructor
  public Dog(String name, int age) {
    this.name = name;
    this.age = age;
  }
  // Methods
  public int getAge() {
    return age;
  }
  public void feedDog() {
    System.out.println("Gave " + name + " " + calcFoodAmount() + " of food.");
  }
  private String calcFoodAmount() {
    if (age < 6) {
      return "3 bowls";
    }
    return "1 bowl";
  }
}
```

```java
public class Dog {
  // Attributes
  public String name;
  private int age;
  // Constructor
  public Dog(String name, int age) {
    this.name = name;
    this.age = age;
  }
  // Methods
  public int getAge() {
    return age;
  }
  public void feedDog() {
    System.out.println("Gave " + name + " " + calcFoodAmount() + " of food.");
  }
  private String calcFoodAmount() {
    if (age < 6) {
      return "3 bowls";
    }
    return "1 bowl";
  }
}
```

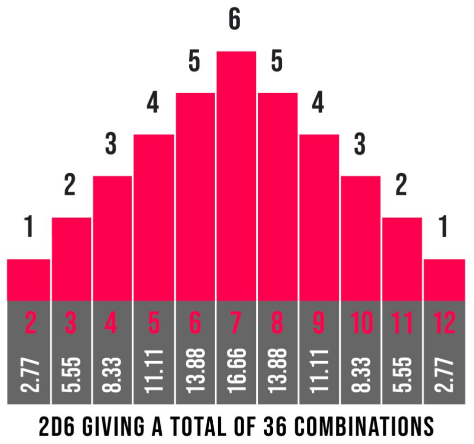
```
public class Dog {
  // Attributes
  public String name;
  private int age;
  // Constructor
  public Dog(String name, int age) {
    this.name = name;
    this.age = age;
  }
  // Methods
  public int getAge() {
    return age;
  }
  public void feedDog() {
    System.out.println("Gave " + name + " " + calcFoodAmount() + " of food.");
  }
  private String calcFoodAmount() {
    if (age < 6) {
      return "3 bowls";
    }
    return "1 bowl";
  }
}
```

Practice

- Complete all the activities for 5.1 on CSAwesome
- RefactorMe replit

In computer programming and software design, **code refactoring** is the process of restructuring existing computer code—changing the *factoring*—without changing its external behavior. Refactoring is intended to improve the design, structure, and/or implementation of the software (its *non-functional* attributes), while preserving its functionality.

https://en.wikipedia.org/wiki/Code_refactoring


2D6 GIVING A TOTAL OF 36 COMBINATIONS