Java Programming Foundations 1

Week 11: Designing Object-Oriented Programs

Overview

- Review of class properties, methods, and constructors
- Discuss OOP, or Object-Oriented Programming
- Practice time: Pokemon fight simulator

Properties

Properties

Properties store the current state for an object. They represent all of the data that makes up a class.

Properties

Class properties are similar to variables but belong to an instance of a class.

Properties

Properties can have different access modifiers, but let's just use `private` for now. More on this later in class.

Properties example

```java
public class Car
{
    private String exteriorColor;
    private String interiorColor;
    private int numberOfWheels;
    private int numberOfDoors;
    private int numberOfSeats;
    private int currentXCoordinate;
    private int currentYCoordinate;
}
```

Methods

Methods

Methods let you define actions that an object can take. They represent the behaviour of a class.

Methods example

```
public class Car
{
    private int currentXCoordinate;
    private int currentYCoordinate;

    public void driveForward()
    {
        this.currentXCoordinate += 1;
    }

    public void driveBackward()
    {
        this.currentXCoordinate -= 1;
    }

    public void driveRight()
    {
        this.currentYCoordinate += 1;
    }

    public void driveLeft()
    {
        this.currentYCoordinate -= 1;
    }
}
```

Getters and setters

Getters and *setters* is the term used to describe methods whose purpose is to either get (return) or set the values of the properties of a class.

Getters and setters example

```java
public class Car
{
    private String exteriorColor;
    private int numberOfWheels;

    public void setExteriorColor(String color)
    {
        this.exteriorColor = color;
    }

    public String getExteriorColor()
    {
        return this.exteriorColor;
    }

    public void setNumberOfWheels(int number)
    {
        this.numberOfWheels = number;
    }

    public int getNumberOfWheels()
    {
        return this.numberOfWheels;
    }
}
```

Constructors

Constructors

Constructors are special methods that run when you instantiate a class using the `new` keyword.

Constructors

Constructors always have the same name as that of the class.

Constructors

Just like any other method, constructors can take arguments.

Constructors

You can overload your constructor, meaning you can have multiple constructors that run according to the arguments that were used when your class was instantiated.

Default constructors

A default constructor is the constructor of a class that does not define any parameters. It's not required, but useful to implement in your classes.

Constructors example

```java
public class Car
{
    private String exteriorColor;
    private String interiorColor;
    private int numberOfWheels;
    private int numberOfDoors;

    Car()
    {
        this.exteriorColor = "Gray";
        this.interiorColor = "Black";
        this.numberOfWheels = 4;
        this.numberOfDoors = 4;
    }

    Car(String exteriorColor, String interiorColor)
    {
        this.exteriorColor = exteriorColor;
        this.interiorColor = interiorColor;
        this.numberOfWheels = 4;
        this.numberOfDoors = 4;
    }

    Car(String exteriorColor, String interiorColor, int numberOfWheels, int numberOfDoors)
    {
        this.exteriorColor = exteriorColor;
        this.interiorColor = interiorColor;
        this.numberOfWheels = numberOfWheels;
        this.numberOfDoors = numberOfDoors;
    }
}
```

OOP

OOP

OOP, or Object-Oriented Programming, is a way of programming where you organize the different parts of your programming in classes.

OOP

OOP lets you encapsulate both the data and the behaviour of an entity in a single class.

Practice time

Practice time

Let's build a Pokemon fight simulator.

Pokemon fight simulator classes

- `Pokemon`, represents a Pokemon. Has a name, health, and an attack value.
- `Trainer`, represents a Pokemon trainer. Has a name and an array of Pokemons.
- `Battle`, our fight simulator. It makes two trainers fight and tells us who won.
- `App`, the class where we make it all happen.

```java
public class Pokemon
{
    private String name;
    private int health;
    private int attack;

    Pokemon(String name, int health, int attack)
    {
        this.name = name;
        this.health = health;
        this.attack = attack;
    }

    public String getName()
    {
        return this.name;
    }

    public int getAttack()
    {
        return this.attack;
    }

    public int getHealth()
    {
        return this.health;
    }

    public void takeDamage(int damage)
    {
        this.health -= damage;
    }
}
```

```java
public class Trainer
{
    private String name;
    private Pokemon[] pokemons;

    Trainer(String name, Pokemon[] pokemons)
    {
        this.name = name;
        this.pokemons = pokemons;
    }

    public String getName()
    {
        return this.name;
    }

    public Pokemon getActivePokemon()
    {
        for (int i = 0; i < this.pokemons.length; i++) {
            if (this.pokemons[i].getHealth() > 0) {
                return this.pokemons[i];
            }
        }

        return null;
    }

    public boolean hasActivePokemon()
    {
        return this.getActivePokemon() != null;
    }
}
```

```java
public class Battle
{
    private Trainer trainer1;
    private Trainer trainer2;

    Battle(Trainer trainer1, Trainer trainer2)
    {
        this.trainer1 = trainer1;
        this.trainer2 = trainer2;
    }

    public Trainer getWinner()
    {
        while (
            this.trainer1.hasActivePokemon() &&
            this.trainer2.hasActivePokemon()
        ) {
            performSingleAttack();
        }

        if (this.trainer1.hasActivePokemon()) {
            return this.trainer1;
        } else if (this.trainer2.hasActivePokemon()) {
            return this.trainer2;
        }

        return null;
    }

    private void performSingleAttack()
    {
        Pokemon pokemon1 = this.trainer1.getActivePokemon();
        Pokemon pokemon2 = this.trainer2.getActivePokemon();

        if (pokemon1 != null && pokemon2 != null) {
            pokemon2.takeDamage(pokemon1.getAttack());
            pokemon1.takeDamage(pokemon2.getAttack());
        }
    }
}
```

```java
public class App
{
    public static void main(String[] args)
    {
        Trainer trainer1 = new Trainer("Marcos", new Pokemon[]{
            new Pokemon("Pikachu", 100, 7),
            new Pokemon("Bulbasaur", 140, 4),
        });

        Trainer trainer2 = new Trainer("Andrea", new Pokemon[]{
            new Pokemon("Squirtle", 140, 5),
            new Pokemon("Charmander", 120, 8),
        });

        Battle battle = new Battle(trainer1, trainer2);
        Trainer winner = battle.getWinner();
        if (winner == null) {
            System.out.println("It was a tie!");
        } else {
            System.out.println("The winner is " + winner.getName());
        }
    }
}
```

Hint for this week's homework

When comparing two `String` values, use the `equals` method.

String comparison example

```java
public class Sample
{
    public static void main(String[] args)
    {
        String name1 = "Marcos";
        String name2 = "Andrea";

        if (name1.equals(name2)) {
            System.out.println("name1 and name2 are equal.");
        } else {
            System.out.println("name1 and name2 are different.");
        }
    }
}
```