OBJECT-ORIENTED JAVA

By Hunter Stevenson

CSCI 5448

Introduction

- Java is a general-purpose, concurrent, class-based, object-oriented language that is specifically designed to have as few implementation dependencies as possible.

- “Write once, run anywhere.”

OOP Concepts

- 5 Main Concepts:
  - Object
  - Class
  - Inheritance
  - Interface
  - Package

Object

- State
  - e.g. Dogs (breed, color, name)
  - Stored in *fields* and exposes behaviors through *methods*.
- Behavior
  - e.g. (bark, eat, sit, sleep)

Object (continued)

- Individual objects within a program benefits:
  1. Modularity
  2. Information-hiding
  3. Reusability
  4. Debugging ease

Encapsulation

- By hiding the internal state and requiring communication to be mediated through an object's method.
- The text goes a little further and states that it "hides data and algorithms from the program to prevent dependencies from occurring."
- Groups related concepts into one item.

Abstraction

- Services and concepts that are provided to solve a problem.
- “What it is offering.”
- Focuses on the essential attributes and behavior.
- The process of abstracting common features from objects, and creating a single interface to complete multiple tasks.
- Separates interface and implementation.

Composition

- Complex systems organized using simpler systems.
- An organized collection of smaller components interacting to achieve a coherent and common behavior.
- 2 Types
  - Association: considers each part as a separate unit.
  - Aggregation: considers the composed part as a single unit.

Class

- Represents a category of things.
- Allows for uniformity within a program.
- In OOP, car is an *instance* of the *class of objects* known as cars.
- A *class* is the blueprint from which individual objects are created.

Class (continued)

```
class Car {
        int speed = 0;
        int gear = 1;

        void shift(int newValue){
                gear = newValue;
        }

        void speedIncrease(int paceup){
                speed = speed + paceup;
        }

        void applyBrake(int pacedown){
                speed = speed - pacedown;
        }

        void printStates() {
                System.out.println("Speed:" +speed+" Gear:"+gear);
        }
}
```

Class (continued)

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

                Car auto1 = new Car();
                Car auto2 = new Car();

                auto1.speedIncrease(15);
                auto1.shift(2);
                auto1.printStates();

                auto2.speedIncrease(15);
                auto2.shift(2);
                auto2.speedIncrease(15);
                auto2.shift(3);
                auto2.printStates();
        }
}
```

Class (continued)

CarExample output:

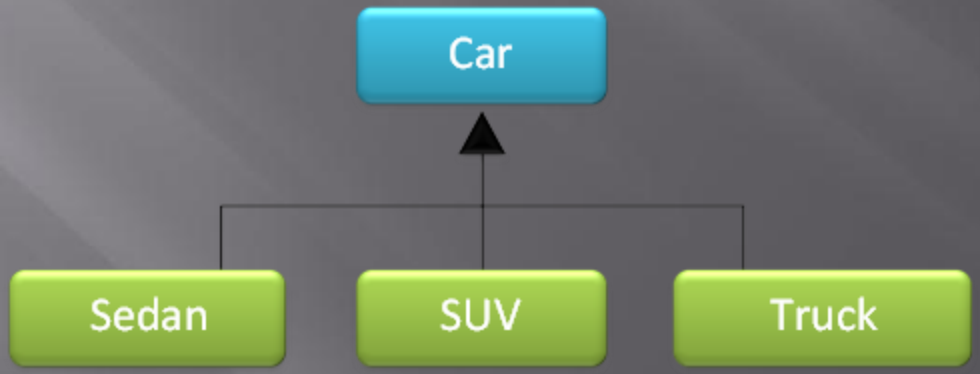
Speed: 15 Gear: 2
Speed: 30 Gear: 3

Polymorphism

- Being able to refer to different derivations of a class in the same way, but getting the behavior appropriate to the derived class that is being referred to.

Inheritance

- OOP allows classes to *inherit* state and behavior from other classes.
- In Java, each class is allowed to have one direct superclass, while each superclass has the potential for an unlimited amount of subclasses.



Inheritance (continued)

```
class sedan extends Car {

}
```

Interface

- Methods form the object's interface with the outside.
- This is the method by which you change the state and initiate the behavior of an object.
- Group of related methods with empty bodies.
- E.g. car key.
  - When inserted into the ignition, the state changes and initiates the behavior to change.

Interface (continued)

```
Interface Car {

        void shift(int newValue);
        void speedIncrease(int paceup);
        void applyBrake(int pacedown);

}
```

Interface (continued)

```
class MaseratiCar implements Car {

}
```

Package

- Namespace that organizes a set of related classes and interfaces.
- Envision a briefcase with separate folders.
- The Java platform provides a large class library (set of packages) and is known as the "Application Programming Interface."

Variables

- *Instance variables* are used for storing the individual states of an object.
- *Class variable* is a variable in a class of which a single copy exists, regardless of the amount of instances of the class.
- *Local variable* is given local scope and is only accessible from the function in which it is declared.
- *Parameters* are always variables, never fields.

Delegation

- Alternative to class inheritance.
- Allows an object composition to be as powerful as inheritance.
- Methods can be delegated by one object to another, however the receiver stays bound to the object doing the delegating.
- In Java, this is a message forwarding concept.

Concurrency

- Represented through threading, synchronization and scheduling.
- Allows for additional complexity and flexibility in software applications.

Events

- Interrupt your program and allow it to respond appropriately.
- Objects can bequeath information and control from themselves to another objects.

Genericity

- Technique to define software components that have more than one interpretation.
- Allows abstraction of data without specifying their exact type.

OOP Advantages

- Bottom-up approach.
- Organized by multiple classes and objects.
- Each class related in a hierarchical manner.
- Views data and function as a single entity.
- Data has importance.
- Solution is problem-domain specific.
- Orchestrated by a delegation of responsibilities.

Executive Summary

- Java as an object-oriented language provides many beneficial concepts to programmers, however there are 5 main concepts to focus on. The first is *Object* and has 2 characteristics: *state* which describes the object and *behavior* which is what the object does once you change its state. The second is *class* which is the blueprint from which objects are created. The third is *inheritance* which allows classes to receive state and behavior from other classes. The fourth is *interface* which is the method by which you change the state and initiate the behavior of an object. The last concept is *package* which simply organizes a set of related classes and interfaces. This presentation covers how each of these concepts will relate to the programming process and how they make programmer's tasks more simplistic!