Unit 3

Advanced Java

Unit 3 Overview

Advanced Java

- **Packages**
- **Define and use Classes**
- **Constructors**
- **Methods**
- **Inheritance**
- **Abstraction**
- **Encapsulation**
- **Polymorphism**
- **Interfaces**
- UML Diagrams
- Collections & Generics
- Exception Handling
- Debugging
- JUnit

```
public static void main(String[] args) {
    System.out.println("Hello World!");
```

Four Fundamental Concepts of OOP

Advanced Java

- **Abstraction** is the process of exposing the essential details of an entity, while ignoring the irrelevant details, to reduce the complexity for the users.

- **Encapsulation** is the process of bundling data and operations on the data together in an entity.

- **Inheritance** is used to derive a new type from an existing type, thereby establishing a parent-child relationship.

- **Polymorphism** lets an entity take on different meanings in different contexts.

Employe
e

*/
public static void main(String[] args) {
System.out.println("Hello World!");

OOP - Believe

Advanced Java

This isn’t going to make perfect sense right away, even after your first OOP assignment. The main thing is that you try to grasp as much as possible about OOP.

You can do this! You can learn OOP!

You can do great things, but only if you believe.

Employe
e

public static void main(String[] args) {
System.out.println("Hello World!");

Stop Me!

Advanced Java

Somebody Stop Me!

Please stop me if you have a question, even it’s from a previous slide. Someone else might have that same question!

Review Part 1

Packages and Classes

Packages

What is a Package?

Packages

- Group of related classes and interfaces
- Similar to folders on your computer
- Used to avoid name conflicts
- Java Platform Packages
  - Application Programming Interface (API)
  - https://docs.oracle.com/en/java/javase/11/docs/api/index.html

```
public static void main(String[] args) {
    System.out.println("Hello World!");
```

Java API

Packages

- java.lang (default package)
    - String
    - System
- java.util
    - Collections
    - Scanner
    - Random

Employe
e

```
*/
public static void main(String[] args) {
    System.out.println("Hello World!");
```

Importing Packages and Classes

Packages

Syntax

```
import package.name.Class;    // Import a single class
import package.name.*;        // Import the whole package
```

Examples

```
import java.util.Scanner;
import java.util.*;
```

Employe
e

*/
public static void main(String[] args) {
System.out.println("Hello World!");

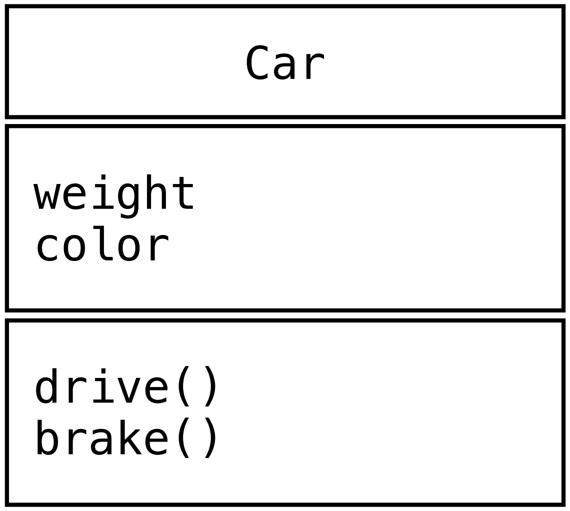
Classes

blueprints

Classes

Blueprint for an object

A class describes what an object knows and what an object does.



public static void main(String[] args) {
System.out.println("Hello World!");

Declaring Classes

Classes

```
class MyClass {
    // field, constructor, and
    // method declarations
}

class MyClass extends MySuperClass implements YourInterface {
    // field, constructor, and
    // method declarations
}
```

Employee

public static void main(String[] args) {
System.out.println("Hello World!");

Declaring Member Variables

Classes

```
public class Car {

    // Member Variables in a class (fields)
    private String weight;
    private String color;

    ...

    // Variables in method declarations (parameters)
   public boolean isSpeeding(int speedLimit, int speed) {

        // Variables in a method or block of code (local variables)
        int difference = speed - speedLimit;
        ...
   }

    ...

}
```

Defining Methods

Classes

Modifiers—such as public, private, and others you will learn about later.

The return type*—the data type of the value returned by the method, or void if the method does not return a value.

The method name*—the rules for field names apply to method names as well, but the convention is a little different.

The parameter list in parenthesis*—a comma-delimited list of input parameters, preceded by their data types, enclosed by parentheses, (). If there are no parameters, you must use empty parentheses.

An exception list—to be discussed later.

The method body*, enclosed between braces—the method's code, including the declaration of local variables, goes here.

```
public int divideTwoNumbers(int number, int anotherNumber) throws ArithmeticException {

    return number / anotherNumber;

}
```

* Required elements of a method declaration

Constructors for Your Classes

Classes

- The purpose of a **constructor** is to initialize the object of a class.
- The purpose of a **method** is to perform a task by executing java code.

```
public class Dog {

    ...

    public Dog() {
    }

    public Dog(String breed, String size, short age, String color) {
        this.breed = breed;
        this.size = size;
        this.age = age;
        this.color = color;
    }

    ...

}

// How to use these constructors
Dog dog1 = new Dog();
Dog dog2 = new Dog(“Aussie”, “Medium”, (short)2, “Black”);
```

Employee
public static void main(String[] args) {
System.out.println("Hello World!");

Passing Information to a Method or Constructor

Classes

- Method or Constructor declaration
  - Declares the **number** and **type** of arguments
- **Parameters** in body are set by the values of the **arguments** passed in
- Parameter Types
  - Any data type; **primitives** or **objects**

```
public Dog(String breed, String size, short age, String color) {
    this.breed = breed;
    this.size = size;
    this.age = age;
    this.color = color;
}

// Example of using a constructor
Dog dog = new Dog("Aussie", "Medium", (short) 2, "Tri-Color");
```

Parameter Names

Passing Information

- Unique to its scope - can’t have same name
    - as another parameter
    - or local variable within the method or constructor
- But the name can be the same name as one of the class fields

```
// Can be the same as a member variable
private String breed;
private String size;

public Dog(String breed, String size) {
   // Local variable can not have the same name
   // String breed = “Aussie”;
}
```

Pass By Value

Parameters

- Passing Primitive Data Type Arguments
    - Example: PassPrimitiveByValue.java
- Passing Reference Data Type Arguments
    - Example: PassObjectReference.java

Employee

```
*/
public static void main(String[] args) {
    System.out.println("Hello World!");
```

Returning a Value from a Method

Classes

A method returns to the code that invoked it when it

- completes all the statements in the method
- reaches a return statement, or
- throws an exception (covered later),

whichever occurs first.

Method Return Type

Classes

```
return;

return returnValue;

public void noReturn() {
}

// a method for computing the area of the rectangle
public int getArea() {
    return width * height;
}

public Bicycle seeWhosFastest(Bicycle myBike, Bicycle yourBike) {
   // code to calculate which bike is faster
   return fastest;
}
```

Employe e

public static void main(String[] args) {
System.out.println("Hello World!");

Summary

Classes

```
// Class Declaration (body between braces)
// Class name preceded by modifiers (public, private, etc.)

public class Dog {

    // fields (contain state information)

    // constructors (initialize new instance)

    // methods (implement behavior)

}

// Create an object from a class by
// using the “new” operator
// and a constructor
Dog myDog = new Dog();
```

Dog Class

Review

Common Characteristics

Dog Class

- age
- breed
- color
- size

Employe
e

```
*/
public static void main(String[] args) {
    System.out.println("Hello World!");
```

Common Behaviors

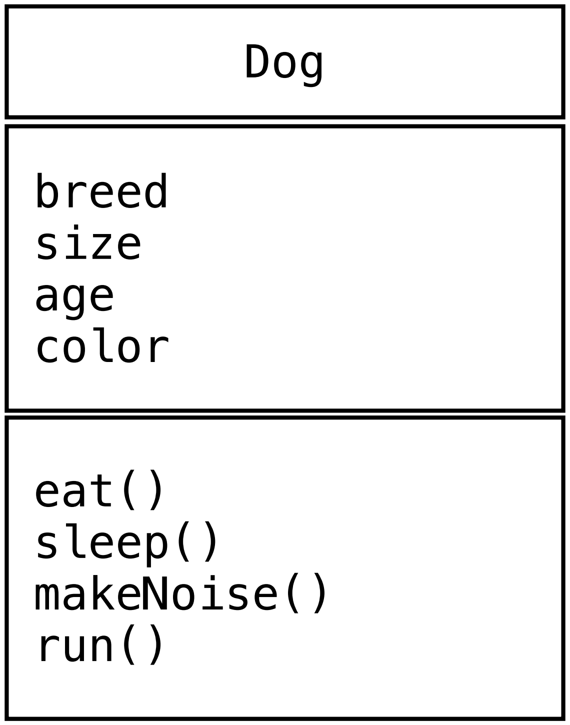
Dog Class

- eat
- sleep
- makeNoise
- run

```
*/
public static void main(String[] args) {
    System.out.println("Hello World!");
```

Class Diagram

Dog Class

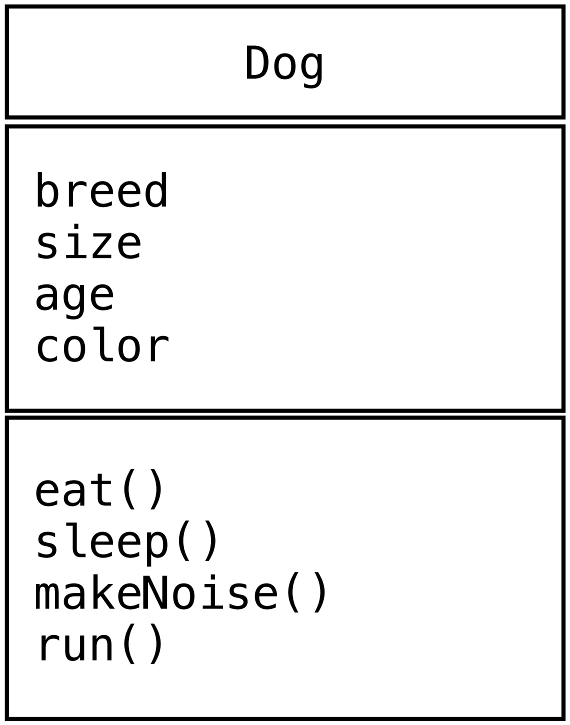


Class Declaration

Dog Class

com.aim.animals.Dog

- add member variables
- add methods



.

[redacted]