The Learning House

427 S 4th Street #300

Louisville KY 40202

JAVA COHORT

01 - Object Basics

Coding Bootcamp

SOFTWARE GUILD

Objectives

- List the 5 basic characteristics object-oriented languages
- Explain the difference between public interface and private implementation
- Understand encapsulation
- Identify intrinsic Java data types
- Understand accessor and mutator methods
- Understand constructors and object instantiation
- Explain the static keyword
- Understand object references
- Understand where storage lives in a running program

Concepts

- Our world is object-oriented — let's talk about lunch...
- Models, metaphors, and abstractions
- We can represent elements in the problem space as objects

Object Orientation

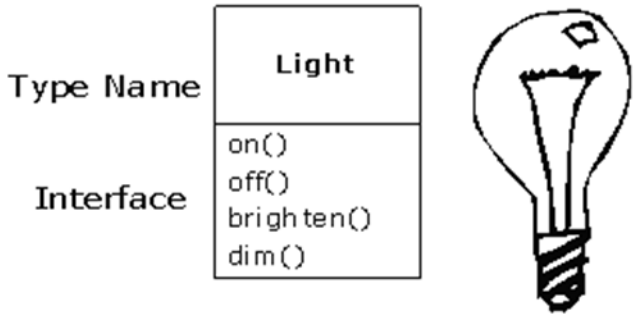
1. Everything is an object.
2. A program is a collection of objects telling each other what to do by sending messages (think method calls).
3. Each object can be made up of other objects.
4. Every object has a type.
5. All objects of a particular type can receive the same messages (i.e. have the same methods).

Types

- Every object has a type.
- It may be one of the intrinsic or built-in types:
  - boolean, byte, char, short, int, long, float, double
- It may be a type defined by a class.
- You are free to create as many new types as you wish.

Public Interfaces, Private Implementations

- Public interface is like a contract that the object must fulfill.
- Implementation is hidden — clients of the object don't need to know how, just what.



Encapsulation

- A good interface has a well-defined area of responsibility and is cohesive.
- Each class should do one thing and do it well.
- It should fully encompass all aspects of its area of responsibility.
- Account example
- Drive-through example

Encapsulation – Related Concepts

- Data Hiding
  - Hiding the internal state of the object
  - Scanner object example
- Delegation
  - Encapsulation dictates that an object fulfills one area of responsibility but it also restricts itself to that area
  - Delegate to things outside area
    - Ex: Scanner, writing to console

Creating New Types

- Java has several native/intrinsic types (int, double, float, etc.)
- New types can be added
- Types are simply new classes:
  - fields
  - methods
- Examples of Java types we've already seen:
  - String
  - Scanner

Classes vs. Objects

- A class is a definition
- An object is the instantiation of that definition
- Examples:
  - Blueprints vs. houses
    - You can’t live in a blueprint
  - The idea of a German shepherd vs. my German shepherd (named Buster)
    - You can’t pet the idea of a German shepherd

Accessors and Mutators

- Use these methods to get and set values for fields on your objects
- Help with data hiding
- Can be used to create read-only fields
- Why go to the trouble?

Controlling Access

Modifier	Class	Package	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	N
<no modifier>	Y	Y	N	N
private	Y	N	N	N

Example

- Dog class

Constructors and “new”

- Create object using new
- Calls the constructor
- Constructors are special methods called upon object creation, used to initialize the object
- Examples:
  - Scanner sc = new Scanner(System.in);
  - String st = new String(“software”);

References and Storage

- **new** hands us back a reference to an object
- An object is created in memory (on the heap)
- We get a reference (or handle) to the object
- An object can have more than one reference

Garbage Collection

- Java is a **managed** language.
- We do not have to explicitly ask for memory for our objects (the **new** operator handles this for us).
- When there are no more references to an object, it is marked for **garbage collection**.
- The **garbage collector** reclaims unused memory.

Example 2

- Dog class with references

Static

- Method or data not tied to any particular instance of the class
- Can be accessed without creating an instance of the class
- Non-static data or methods must be associated with a particular instance of the class

Example 3

- Dog tracker
- String
- Math

Dot operator

- Used whenever you want to access a method or member of a class
- Can be used for both static and non-static members
- Example:
  - Math.abs(..) (static)
  - System.out.println(); (static)
  - myDog.bark(); (non-static)

“this” keyword

- **this** refers to the instance of the object in which the code is currently executing
- Used with dot operator
- Examples:
  - Getters/setters
  - Other methods
  - Constructor

Nested Classes

- Java allows you to define classes inside of other classes.
- Two categories:
  - Static: referred to as **static nested classes**
  - Non-static: referred to as **inner classes**

  NOTE: By definition, there is no such thing as a static inner class!
- Because inner classes are associated with a particular instance of the surrounding class, they cannot have any static members of their own.

Why Use Nested Classes?

- Organization — helper classes that are only useful to the class in which they are nested.
- Encapsulation — the nested class can be hidden but can access private members of the surrounding class.
- Readability — the class definitions are close together.
- Note: A static nested class has the same access to the surrounding class as any other top-level class.

Shadowing

- A declaration of a variable that has the same name as a variable in the enclosing scope will shadow the declaration of the enclosing scope.
- You can’t use the shadowed variable name by itself — instead, you must include some reference to its scope.
- Let’s look at an example of an inner class and some shadowed variables...

Local Classes

- Java allows you to define classes inside any block — typically, that block will be a method.
- A local class can access members of the enclosing class.
- A local class also has access to the local variables of its surrounding block *if those variables are declared final (all versions) or are effectively final (Java 8 only)*.
- Local classes cannot have any static members.

Anonymous Classes

- Can make code more concise
- Can be used in a similar manner as lambdas or closures from other languages
- Syntax is a bit clunky
- Essentially, you declare and instantiate the class at the same time
- Let’s look at an example of an anonymous inner class...

Lambdas

- Java 8 has added a functional feature called Lambdas
- Lambdas allow you to express an instance of a single method class in a more compact manner