JAVA BOOTCAMP

OBJECT ORIENTED PROGRAMMING

CLASSES AND OBJECTS

TOPICS:

- Creating classes
- Creating and using objects

REVIEW

REVIEW

OBJECT-ORIENTED PROGRAMMING

Explain some of the foundational concepts that we've covered already. How do they interact?

OBJECT-ORIENTED PROGRAMMING

OBJECT-ORIENTED PROGRAMMING

OBJECT-ORIENTED CONCEPTS

- Abstraction
- Encapsulation
- Inheritance
- Polymorphism

OBJECT-ORIENTED PROGRAMMING

OBJECT-ORIENTED CONCEPTS

- Abstraction: Hide the details and only show functionality.
- Example: When you do a phone call, you don’t need to know the internal process of how the wires carry your voice.

OBJECT-ORIENTED PROGRAMMING

OBJECT-ORIENTED CONCEPTS

- Encapsulation: Wrapping related functionality and data together as one unit.
- Example: Using classes to define an employee.

OBJECT-ORIENTED PROGRAMMING

OBJECT-ORIENTED CONCEPTS

- Inheritance: A class acquiring properties and behaviors of parent class.
- Provides code reusability, and used to achieve the forth concept, Polymorphism.

OBJECT-ORIENTED PROGRAMMING

OBJECT-ORIENTED CONCEPTS

- Polymorphism: Same task can be done in different ways depending on the object's type. Polymorphism is also the ability of an object to take on many forms.
- Example:

  An Employee is a Person

  A Circle is a Shape

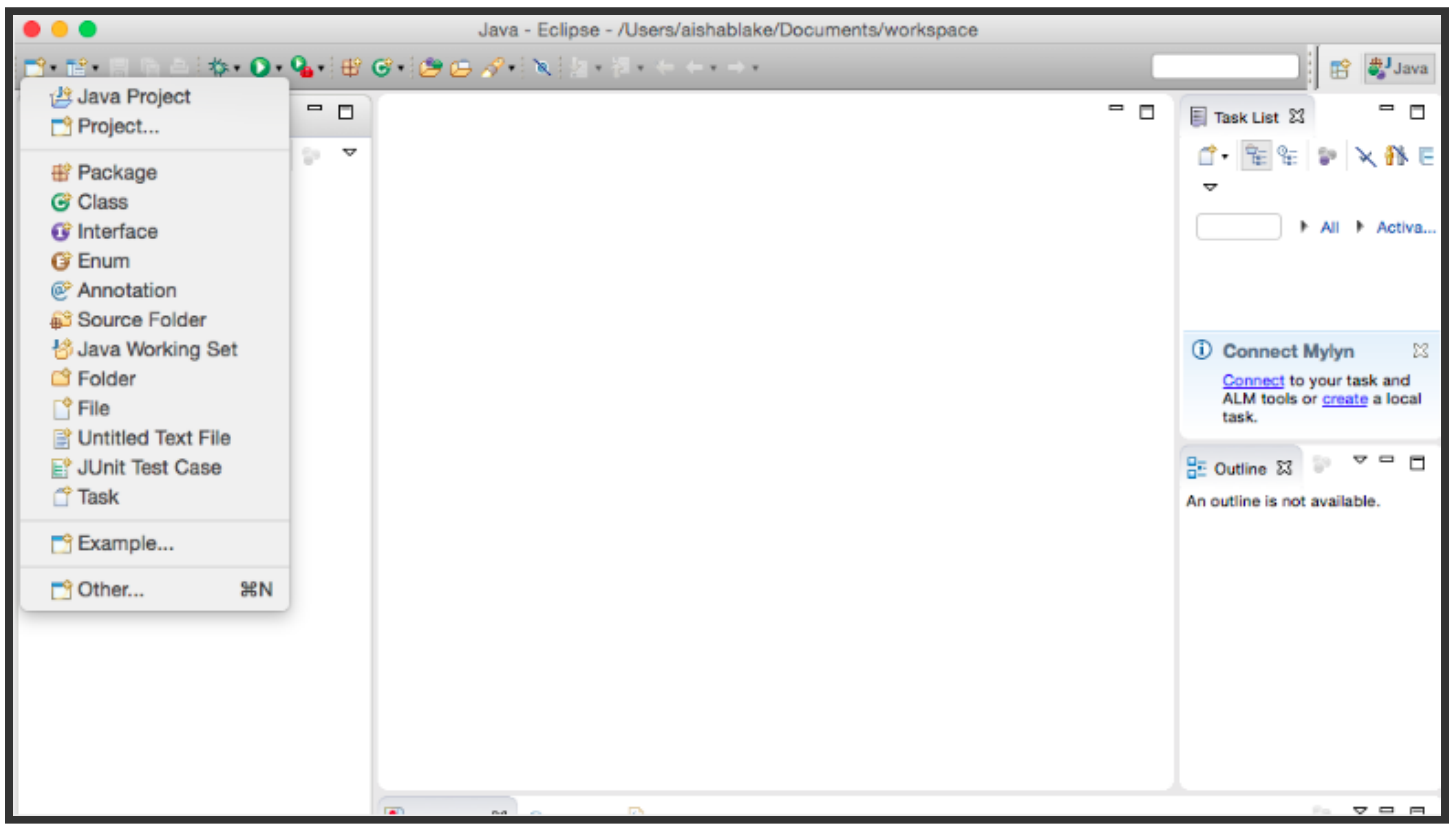
OBJECT-ORIENTED PROGRAMMING

CLASS VS. OBJECT

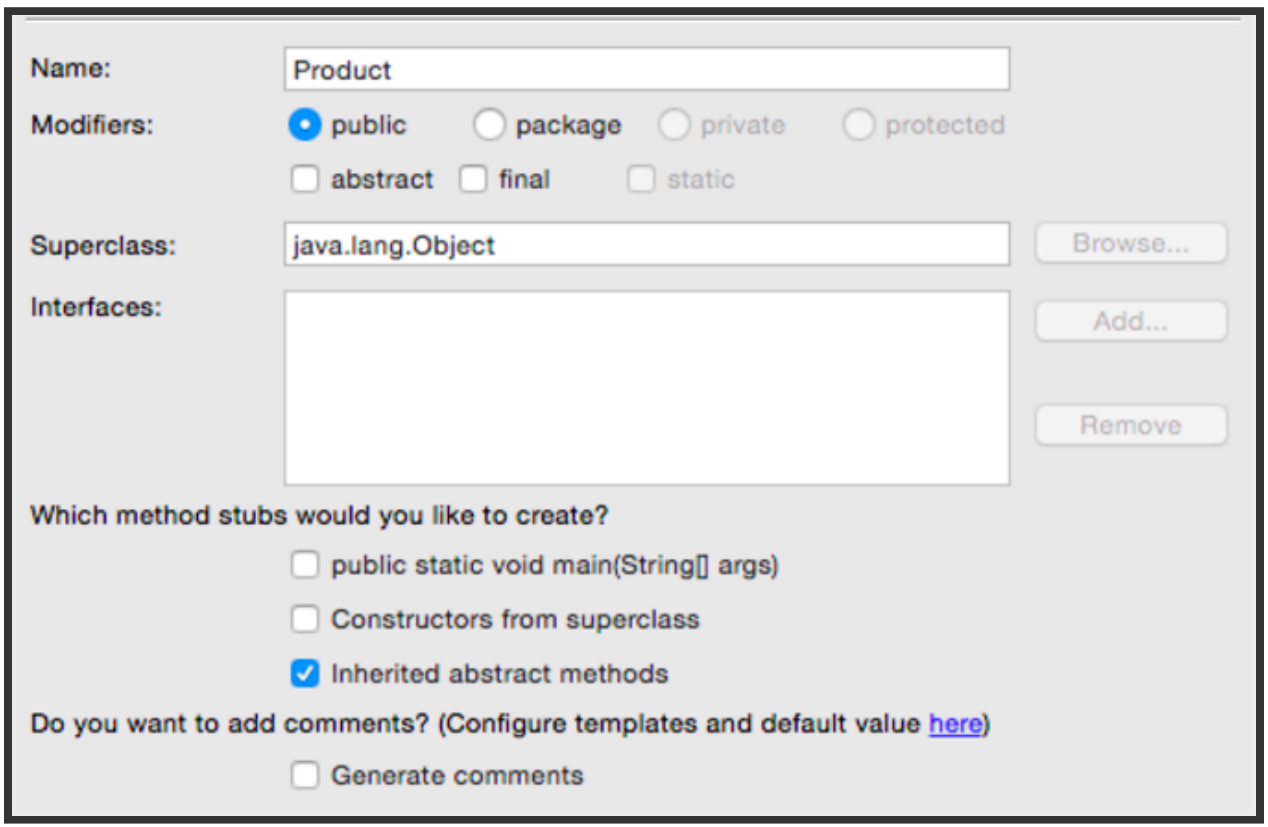
- A class is a template or a blue print that describes the supported behavior and state of all objects of its type.
- An object is an instance of a class.

CREATING A NEW CLASS IN ECLIPSE

CREATING A NEW CLASS IN ECLIPSE



CREATING A NEW CLASS IN ECLIPSE



Input a capitalized name in the appropriate box

INSTANCE VARIABLES

INSTANCE VARIABLES

DESCRIPTION

- An instance variable may be a primitive data type, an object created from a java class such as the String class, or an object created from a user-defined class such as the Product class.
- Created when the object is created, in contrast to static variables.

INSTANCE VARIABLES

DESCRIPTION

- To prevent other classes from accessing instance variable, use the *private* keyword to declare the as *private*.
- We can declare the instance variable for a class anywhere outside the constructors and methods of the class.

INSTANCE VARIABLES

SYNTAX

```
public|private|protected Type variableName;
```

INSTANCE VARIABLES

EXAMPLES

```
public double price;
private int quantity;
private String code;
private Product product;
```

INSTANCE VARIABLES

CONTEXT

```
public class Product
{
    //common to code instance variables here
    private String code;
    private String description;
    private double price;
    //the constructors and methods of the class
    …
    //also possible to code instance variables here
    private int test;
}
```

CONSTRUCTORS

CONSTRUCTORS

DESCRIPTION

- A method that is used to initialize the state of an object.
- A constructor must use the same name and capitalization as the name of the class.

CONSTRUCTORS

DESCRIPTION

- In the absence of a constructor Java will create a default constructor that initializes all numeric types to 0, all boolean types to false, and all objects to null.
- The name of the class combined with the parameter list forms the signature of the constructor. Each constructor must have a unique signature.

CONSTRUCTORS

SYNTAX

```
public ClassName([parameterList])
{
        //the statements of the constructor
}
```

CONSTRUCTORS

EXAMPLE

```
public Product(String code, String description, double price)
{
        Mycode = code;
        Mydescription = description;
        Myprice = price;
}
```

THE THIS KEYWORD

DESCRIPTION

- Can be used to refer to instance methods and data defined inside the class.
- Since Java implicitly uses the this keyword for instance variables and methods, we don’t need to explicitly code it unless a parameter has the same name as an instance variable.
- If you use the this keyword to call another constructor, the statement must be the first statement in the constructor.

THE THIS KEY

SYNTAX

```
this.variableName
        //refers to an instance variable of the current object
```

EXAMPLE

```
public void setX(int x)
{
        this.x=x;
}
```

RECAP

RECAP

WHAT YOU SHOULD KNOW AT THIS POINT.

- What is OOP
- OO concepts
- What are classes
- What are objects
- Difference between classes and objects
- How to create classes and objects
- Know how to define instance fields
- Define and use constructors
- Using this keyword

INHERITANCE AND POLYMORPHISM

INHERITANCE AND POLYMORPHISM

TOPICS

- Intro to inheritance
- Basics of working with inheritance
- Polymorphism
- Final keyword

INHERITANCE

INHERITANCE

WHAT IS INHERITANCE?

Inheritance allows us to create new classes based on existing ones.

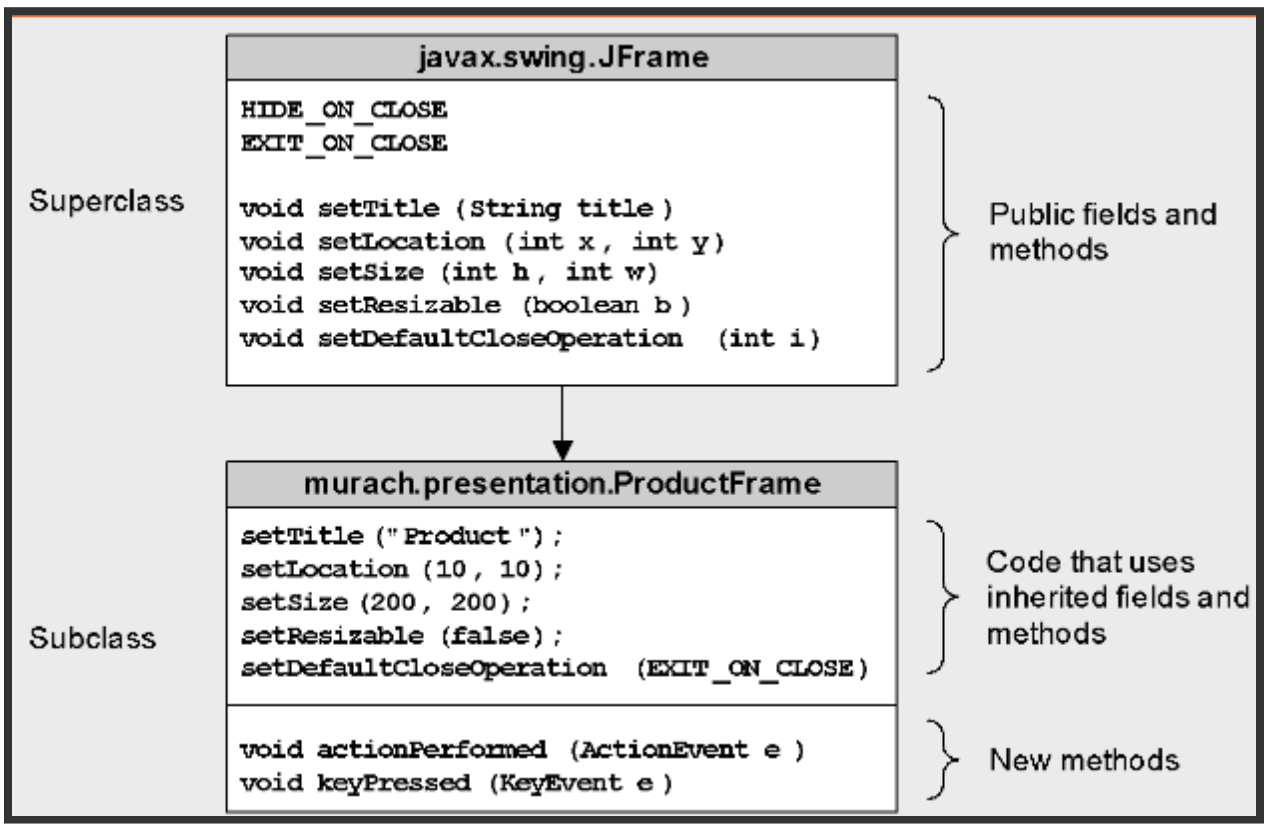
A new class (*subclass*, *derived class*, *child class*) inherits the fields, constructors, and methods of the class it is based on (*superclass*, *base class*, *parent class*)

INHERITANCE

EXTENDING CLASSES

A subclass can *extend* the superclass by adding new fields, constructors, and methods. It can also *override* a method from the superclass with its own version of the method.

HOW INHERITANCE WORKS



INHERITANCE

USING INHERITANCE

- Create generic superclasses that implement common elements of related subclasses.
- Create classes that inherit from classes that are defined by the Java API.

SUB- AND SUPERCLASSES

CREATING A SUPERCLASS

ACCESS MODIFIERS

Access modifiers specify the accessibility of the members declared by a class.

Keyword	Description
private	Available within the current class
public	Available to classes in all packages
protected	Available to in the same package and to subclasses outside the packag
default	Only accessible within the package

CREATING A SUBCLASS

SYNTAX

Declaring a sublcass

```
public class SubclassName extends SuperclassName { }
```

CREATING A SUBCLASS

DESCRIPTION

- We can directly access fields that have public or protected access in the superclass from the subclass.
- We can override methods in the superclass by coding methods in the subclass that have the same signatures as methods in the superclass.
- We can use the *super* keyword to call a constructor or method of the superclass.

WORKING WITH INHERITANCE

WORKING WITH INHERITANCE

CASTING OBJECTS

- Java can implicitly cast a subclass to superclass, but we must explicitly cast a superclass when we need to reference one of its subclasses.
- Casting affects the methods that are available from an object.

WORKNG WITH INHERITANCE

GETTING INFORMATION ABOUT AN OBJECT'S TYPE.

- Every object has a getClass() method that returns a Class object that corresponds to the object's type.
- There are over 90 properties and methods of the Class class!
- We can use the instanceof operator to check if an object if an object is an instance of a particular class.

POLYMORPHISM

WHAT IS POLYMORPHISM?

- *Polymorphism* is a feature of inheritance that allows us to treat object of different subclasses that are derived from the same superclass as if they had the type of the superclass.
- Example.

THE FINAL KEYWORD

THE FINAL KEYWORD

DESCRIPTION

- We can prevent a class from being inherited by using the *final* keyword.
- A *final method* cannot be overridden by a subclass. All methods within final classes are final methods.
- The final keyword may also be used to prevent a method from assigning a new value to a parameter.

THE FINAL KEYWORD

EXAMPLES

```
public final class Book extends Product {
    // all methods in the class are automatically final
}

public final String getVersion() {
    return version;
}

public void setVersion(final String version) {
    // version = "new value"; // not allowed
    this.version = version;
}
```

RECAP

RECAP

WHAT YOU SHOULD KNOW AT THIS POINT

- What is Inheritance.
- Importance of Inheritance.
- How Inheritance works.
- What is polymorphism.
- Why we need polymorphism.
- Using the final keyword.

INTERFACES AND ABSTRACT CLASSES

INTERFACES

TOPICS

- Abstract classes
- Introduction to interfaces
- Working with interfaces and abstract classes

THE ABSTRACT KEYWORD

THE ABSTRACT KEYWORD

DESCRIPTION

- An abstract class can be inherited by other classes, but not used to create an object.
- An abstract method is also created using the abstract keyword. Abstract methods have no body. They cannot have private access.
- An abstract class does not have to contain abstract methods, but any class that does contain abstract methods must be declared as abstract.

THE ABSTRACT KEWORD

EXAMPLE

```
public abstract class Product {
    private String code;
    private String description;
    private double price;
    // regular constructors and methods for instance variables
    @Override
    public String toString() {
        return "Code:         " + code + "\n" + "Description: " + description + "\n" +
               "Price:        " + this.getFormattedPrice() + "\n";
    }
    public abstract String getDisplayText(); // an abstract method
}
```

INTERFACES VS. ABSTRACT CLASSES

AN INTRODUCTION TO INTERFACES

An interface is a special type of coding element that provides many of the advantages of multiple inheritance. An interface defines a set of public methods that can be implemented by a class.

AN INTRODUCTION TO INTERFACES

A SIMPLE INTERFACE

The code for an interface uses the interface keyword instead of the class keyword and contains only abstract methods. A class that implements an interface must provide an implementation for each method defined by the interface.

INTERFACES VS. ABSTRACT CLASSES

ADVANTAGES OF AN ABSTRACT CLASS

- Can use instance as well as static variables and constants
- Can define regular methods that contain code and abstract methods that don’t contain code
- Can define static methods

INTERFACES VS. ABSTRACT CLASSES

ADVANTAGES OF AN INTERFACE

- A class can directly implement multiple interfaces
- Any object created from a class that implements an interface can be used wherever the interface is accepted

INTERFACES VS. ABSTRACT CLASSES

Abstract class	Interface
Variables	Static constants
Constants	
Static variables	
Static constants	
Methods	Abstract methods
Static methods	
Abstract methods	

INTERFACES VS. ABSTRACT CLASSES

A PRINTABLE INTERFACE

```
public interface Printable
{
        void print();
}
```

INTERFACES VS. ABSTRACT CLASSES

A PRINTABLE ABSTRACT CLASS

```
public abstract class Printable
{
        public abstract void print();
}
```

HOW TO WORK WITH INTERFACES

HOW TO WORK WITH INTERFACES

SYNTAX

```
public interface InterfaceName
{
        type CONSTANT_NAME = value;                          // declares a field
        returnType methodName([parameterList]);          // declares a method
}
```

HOW TO WORK WITH INTERFACES

DESCRIPTION

- All methods are automatically declared public and abstract
- All fields are automatically declared public, static, and final
- Interface methods can’t be static

HOW TO WORK WITH INTERFACES

AN INTERFACE THAT DEFINES THREE METHODS

```
public interface ProductWriter
{
        boolean addProduct(Product p);
        boolean updateProduct(Product p);
        boolean deleteProduct(Product p);
}
```

HOW TO WORK WITH INTERFACES

SYNTAX

```
public class ClassName implements Interface1[, Interface2]…{ }
```

DESCRIPTION

If you forget to implement a method that’s defined by an interface that you’re implementing, the class won’t compile.

HOW TO WORK WITH INTERFACES

HOW TO INHERIT A CLASS AND IMPLEMENT AN INTERFACE

You can code a class that inherits another class and implements an interface. To do that, the subclass uses the extends keyword to indicate that it inherits the superclass. Then, it uses the implements keyword to indicate that it inherits the interface.

HOW TO WORK WITH INTERFACES

SYNTAX

```
public class SubclassName extends SuperclassName
                          implements Interface1[, Interface2] … { }
```

HOW TO WORK WITH INTERFACES

DESCRIPTION

- A class can inherit another class and also implement one or more interfaces
- If a class inherits another class that implements an interface, the subclass automatically implements the interface so coding the implements keyword is optional.

HOW TO WORK WITH INTERFACES

HOW TO USE INHERITANCE WITH INTERFACES

When an interface inherits other interfaces, any class that implements that interface must implement all of the methods declared by that interface and the inherited interfaces.

If it doesn't, the class must be declared as abstract so that no objects can be created from it

HOW TO WORK WITH INTERFACES

HOW TO USE INHERITANCE WITH INTERFACES

When a class implements an interface that inherits other interfaces, it can use any of the constants stored in the interface or any of its inherited interfaces.

When a class implements an interface that inherits other interfaces, you can use an object created from that class anywhere any of interfaces in the

inheritance hierarchy.

HOW TO WORK WITH INTERFACES

SYNTAX

```
public interface InterfaceName extends InterfaceName1[, InterfaceName2]…
{
        // the constants and methods of the interface
}
```

HOW TO WORK WITH INTERFACES

DESCRIPTION

- An interface can inherit one or more interfaces by specifying the inherited interfaces in an extends clause
- An interface can’t inherit a class

HOW TO WORK WITH INTERFACES

DESCRIPTION

- A class that implements an interface must implement all the methods declared by the interface as well as all the methods declared by any inherited interfaces unless the class is defined as abstract.
- A class that implements an interface can use any of the constants declared in the interface as well as any constants declared by any inherited

interfaces.

HOW TO WORK WITH INTERFACES

A PRODUCT READER INTERFACE

```
public interface ProductReader
{
        Product getProduct(String code);
        String getProductsString();
}
```

HOW TO WORK WITH INTERFACES

A PRODUCTDAO INTERFACE THAT INHERITS THREE INTERFACES

```
public interface ProductDAO
        extends ProductReader, ProductWriter, ProductConstants
{
}
```

RECAP

RECAP

WHAT YOU SHOULD KNOW AT THIS POINT

- What are abstract classes and why we use them.
- How to code and use abstract classes.
- What are interfaces and why we use them.
- How to code and work with interfaces.
- Difference between abstract classes and interfaces.