

Java Programming

Language

Inheritance

Interfaces

Java

- Class-based, object-oriented design
 - C++ syntax
 - Strongly typed
 - Manages memory, garbage collection
- Extensive class libraries – **robust**
- Primary language for Android – **mobile + desktop**
- Java Virtual Machine (JVM) – **cross platform**

- Open Source
 - Designed by James Gosling 
 - released by Sun Microsystems in 1995
 - Made open source under GNU GPL in 2007
 - Sun and Java acquired by Oracle in 2010



(Almost) Everything is a Class

- **Classes** and objects are core constructs
- OO features: polymorphism, encapsulation, inheritance, ...
- Static member variables and methods
- Resembles C++ on the surface, but really not the same
 - No pointers, all objects are references
 - No type ambiguity; classes resolved at runtime
 - No destructor (due to garbage collector)
 - No multiple inheritance (single only, but with class **Interfaces**)

Java Class Library

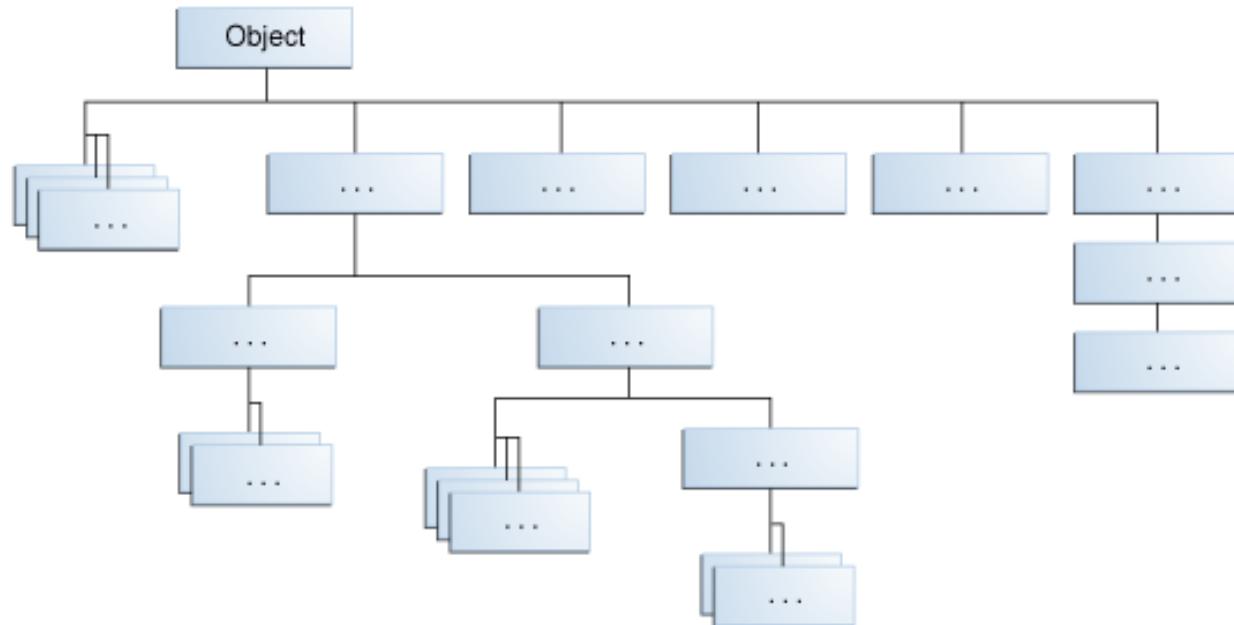
Classes are grouped into "packages"

- **package** keyword to assign source to a package
 - Typically, a package is a subdirectory
 - e.g. "graphics" package is in subdirectory of the same name
- **import** keyword to include a class from a different package
 - This is how you include bundled Java libraries.

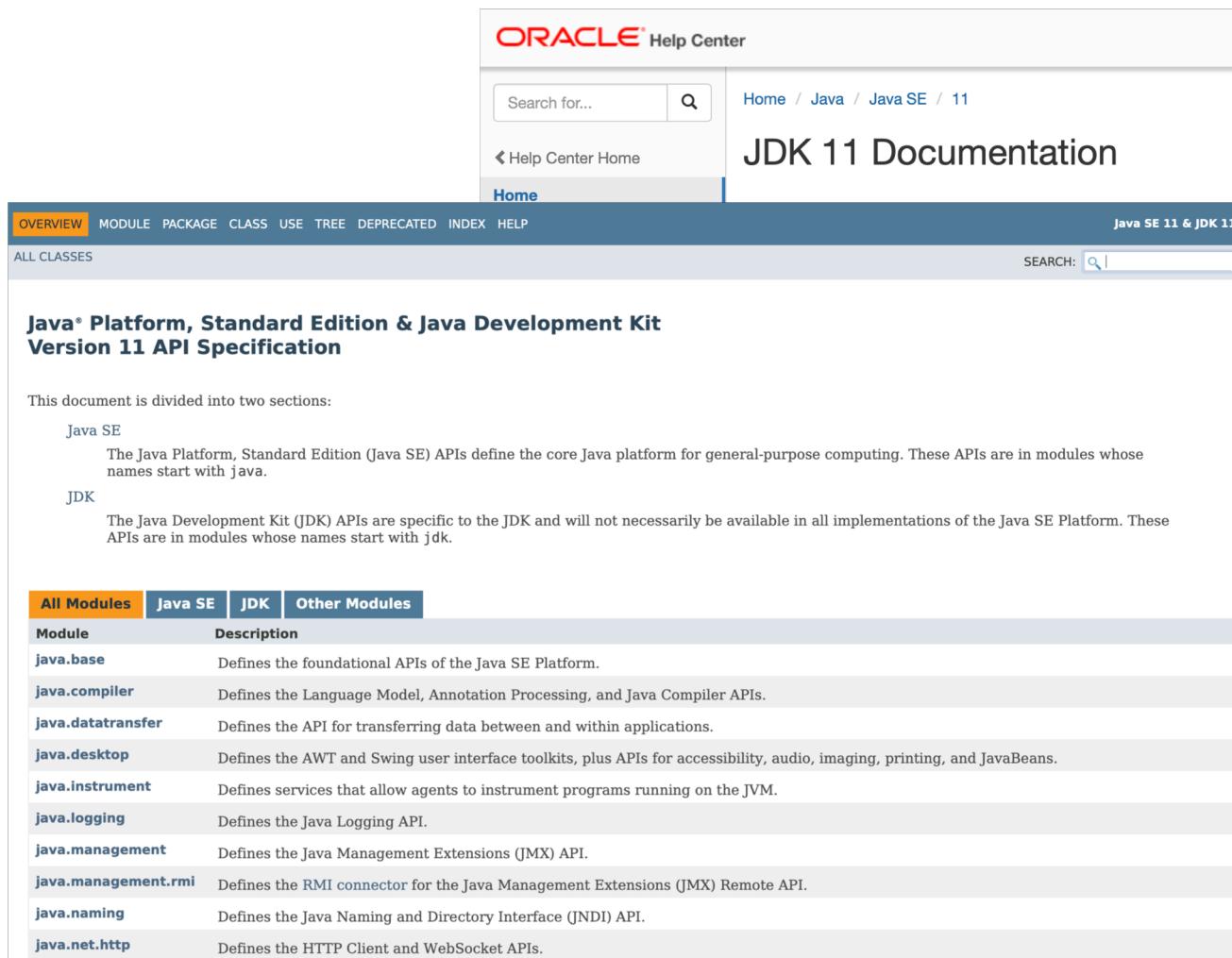
For your assignments, don't worry about putting your code in a package! You can put everything in a single directory (e.g. assignments/a2).

Java Class Hierarchy

- All classes (implicitly) derive from Object class (in java.lang)
 - has methods like `clone()`, `toString()`, `finalize()`
- All classes inherit these basic behaviours



Class Hierarchy & Documentation



The screenshot shows the Oracle Help Center interface for the Java SE 11 & JDK 11 documentation. The top navigation bar includes a search bar, a sign-in link, and a breadcrumb trail: Home / Java / Java SE / 11. Below the navigation is a large circular icon with a coffee cup. The main content area is titled "JDK 11 Documentation". A sidebar on the left lists "OVERVIEW", "MODULE", "PACKAGE", "CLASS", "USE", "TREE", "DEPRECATED", "INDEX", and "HELP". The "OVERVIEW" tab is selected. A "SEARCH:" field is also present. The central content area displays the "Java® Platform, Standard Edition & Java Development Kit Version 11 API Specification". It states that the document is divided into two sections: Java SE and JDK. The Java SE section describes the core Java platform APIs (names starting with java) and the Java Development Kit (JDK) section describes APIs specific to the JDK (names starting with jdk). Below this, a table lists various Java modules with their descriptions:

Module	Description
java.base	Defines the foundational APIs of the Java SE Platform.
java.compiler	Defines the Language Model, Annotation Processing, and Java Compiler APIs.
java.datatransfer	Defines the API for transferring data between and within applications.
java.desktop	Defines the AWT and Swing user interface toolkits, plus APIs for accessibility, audio, imaging, printing, and JavaBeans.
java.instrument	Defines services that allow agents to instrument programs running on the JVM.
java.logging	Defines the Java Logging API.
java.management	Defines the Java Management Extensions (JMX) API.
java.management.rmi	Defines the RMI connector for the Java Management Extensions (JMX) Remote API.
java.naming	Defines the Java Naming and Directory Interface (JNDI) API.
java.net.http	Defines the HTTP Client and WebSocket APIs.

On the right side, there are three panels: "Tools" (with links to Tools Reference, JShell User's Guide, and Javadoc Guide), "Specifications" (with a link to API Documentation), and a large empty space.

<https://docs.oracle.com/en/java/javase>

Common Classes/Packages

Package	Classes (Examples)	Description
java.awt	Color, Graphics, Font, Graphics2D, event.	Contains all of the classes for creating user interfaces and for painting graphics and images.
javax.swing	JFrame, JButton, JList, JToolbar	Provides a set of "lightweight" (all-Java language) components that works the same on all platforms.
java.io	File, FileReader, FileWriter, InputStream	Provides for system input and output through data streams, serialization and the file system.
java.lang	Boolean, Integer, String, System, Thread, Math	Provides classes that are fundamental to the design of the Java programming language.
java.util	ArrayList, HashMap, Observable	Contains the collections framework, legacy collection classes, event model,...

Hello Java

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

class definition

static main

constructor

Things that stand out

- Top-level class definition
- Static main method is scoped within the class
- System package is used for basic IO

class

fields

constructor

methods

main

```
class Bicycle {  
    String owner = null;  
    int speed = 0;  
    int gear = 1;  
  
    // constructor  
    Bicycle() { }  
    Bicycle(String name) { owner = name; }  
  
    // methods  
    void changeSpeed(int newValue) { speed = newValue; }  
    void changeGear(int newValue) { gear = newValue; }  
    int getSpeed() { return speed; }  
    int getGear() { return gear; }  
  
    // static entry point - main method  
    public static void main(String[] args) {  
  
        Bicycle adultBike = new Bicycle("Jeff");  
        adultBike.changeSpeed(20);  
        System.out.println("speed=" + adultBike.getSpeed());  
  
        Bicycle kidsBike = new Bicycle("Austin");  
        kidsBike.changeSpeed(15);  
        System.out.println("speed=" + kidsBike.getSpeed());  
    }  
}
```

Object passing

Parameters are passed by value. However, this isn't always obvious.

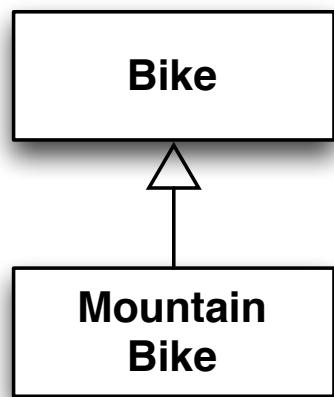
- Primitive types (**int**, **float**, etc.) are allocated on the stack and always passed by value.
- Objects are allocated on the heap, and the *address* is passed-by-value
 - Java uses pass-by-reference semantics, so it *appears* as-if you're passing by reference.
 - Practically, you can treat Java as passing objects by reference.
- There are no pointer semantics in Java
 - i.e. no *****, no **&**, no **out**, no **ref**

both refer to same
memory on the
heap

```
Bicycle my_bike = new Bicycle();  
Bicycle kids_bike = my_bike;
```

Inheritance

- Inherit some methods or fields from a base class (“is a”)
- Very common in Java to extend and override other classes
- Example:
 - “Mountain Bike” is-a “Bike”
 - Mountain bike inherits speed and gear fields
 - Mountain bike defines addition field for suspension type



container class

abstract inner base
class

inner derived class

“ Meow! ”
“ Woof! ”

```
public class Animals1 {  
  
    // base class  
    abstract class Pet {  
        abstract String talk();  
    }  
  
    // derived classes  
    class Cat extends Pet {  
        String talk() { return "Meow!"; }  
    }  
  
    class Dog extends Pet {  
        String talk() { return "Woof!"; }  
    }  
  
    // container class methods  
    Animals1() {  
        speak(new Cat());  
        speak(new Dog());  
    }  
  
    void speak(Pet a) {  
        System.out.println( a.talk() );  
    }  
  
    // static main methods -- entry point  
    Run | Debug  
    public static void main(String[] args) {  
        new Animals1();  
    }  
}
```

Animals1.java

Interfaces

- Java only supports single (class) inheritance
- However, to specify complex class behavior, you can use interfaces
- An interface represents an API (i.e. it defines functionality)
 - It represents a set of methods a class must have
 - You can't instantiate an interface
 - Essentially, it's a pure abstract class
- Classes can choose to *implement* an interface
 - A class that implements an interface must implement ***all*** of the methods in the interface
 - A class can implement multiple interfaces, representing different behaviours that the class wants to support (e.g. you might have interfaces indicating that something printable and loggable).

interface

implementations

The interface Pet is
like a type

```
public class Animals2 {  
  
    // interface  
    interface Pet {  
        String talk();  
    }  
  
    // inner classes  
    class Cat implements Pet {  
        public String talk() { return "Meow!"; }  
    }  
  
    class Dog implements Pet {  
        public String talk() { return "Woof!"; }  
    }  
  
    // container class methods  
    Animals2() {  
        speak(new Cat());  
        speak(new Dog());  
    }  
  
    void speak(Pet a) {  
        System.out.println( a.talk() );  
    }  
  
    // static main methods -- entry point  
    Run | Debug  
    public static void main(String[] args) {  
        new Animals2();  
    }  
}
```

Animals2.java

base class

```
// base class
abstract class Bike {
    int wheels = 0;
    int speed = 0;

    void setWheels(int val) { wheels = val; }
    void setSpeed(int val) { speed = val; }
    void show() {
        System.out.println("wheels = " + wheels);
        System.out.println("speed = " + speed);
    }
}
```

interface

```
// interface for ANYTHING driveable
// could be applied to car, scooter etc.
interface Driveable {
    void accelerate();
    void brake();
}
```

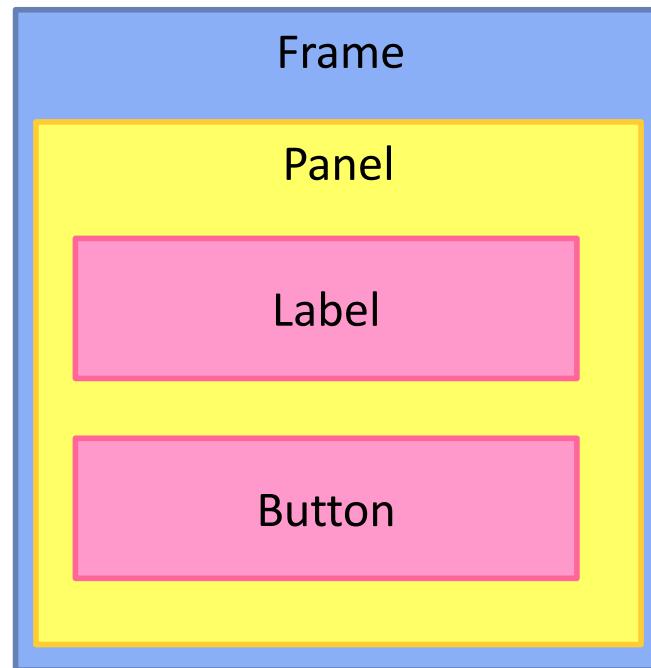
derived class

```
// derived two-wheel bike
class Bicycle extends Bike implements Driveable {
```

Bikes2.java

How to Build a Simple User Interface

- Create a top-level application window (JFrame or JDialog).
- Add components to this window.
 - Typically, you create a smaller container (like a JPanel) and add components to the panel.



Creating a Window

BasicForm1.java

```
import javax.swing.*;  
  
// Create a simple form  
public class BasicForm1 {  
    public static void main(String[] args) {  
        // create a window  
        JFrame frame = new JFrame("Layout Demo");  
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
  
        // create a panel and add components  
        // all Swing components are types of JComponent  
        JPanel panel = new JPanel();  
        JButton button = new JButton("Ok");  
        panel.add(button);  
  
        // add panel to the window  
        frame.add(panel);  
  
        // set window behaviour and display it  
        frame.setResizable(false);  
        frame.setSize(200, 200);  
        // frame.pack();  
        frame.setVisible(true);  
    }  
}
```