

Java GUI Programming.

A quick-start guide to building Swing applications.
With examples and pictures.



Introduction

Background

Design goals

Background

- Designed by James Gosling
- Released by Sun Microsystems in 1995.
 - Originally a proprietary license, but made open source under GNU GPL in 2007.
 - Sun and Java acquired by Oracle in 2010.
- General-purpose, portable language.
 - Cross-platform, and able to scale from small devices to large applications.
 - Dynamic, interpreted*
- Broadly adopted, and extremely successful.
 - “Java is TIOBE's Programming Language of 2015!” (www.tiobe.com)

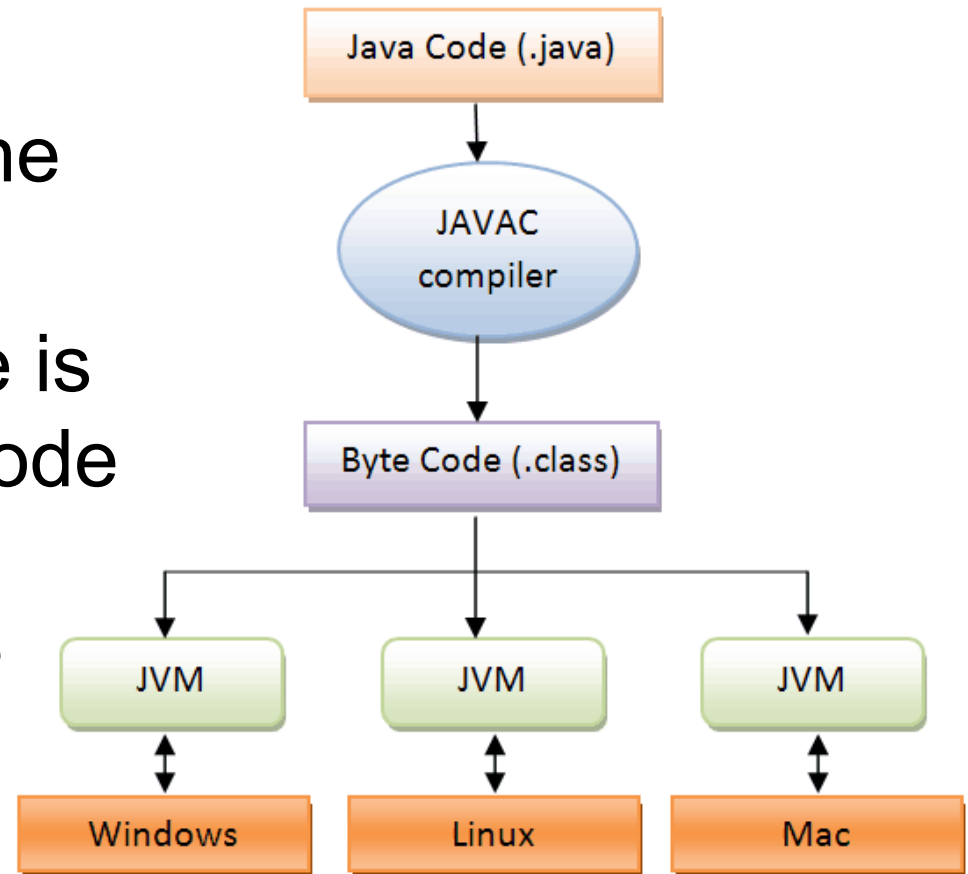
Dec 2015	Dec 2014	Change	Programming Language	Ratings	Change
1	2	⬆	Java	20.973%	+6.01%
2	1	⬇	C	16.460%	-1.13%
3	4	⬆	C++	5.943%	-0.16%
4	8	⬆	Python	4.429%	+2.14%
5	5		C#	4.114%	-0.21%
6	6		PHP	2.792%	+0.05%
7	9	⬆	Visual Basic .NET	2.390%	+0.16%
8	7	⬇	JavaScript	2.363%	-0.07%
9	10	⬆	Perl	2.209%	+0.38%
10	18	⬆	Ruby	2.061%	+1.08%
11	32	⬆	Assembly language	1.926%	+1.40%
12	11	⬇	Visual Basic	1.654%	-0.15%
13	16	⬆	Delphi/Object Pascal	1.639%	+0.52%
14	17	⬆	Swift	1.405%	+0.34%
15	3	⬇	Objective-C	1.357%	-7.77%

Object Oriented

- C++ syntax
 - Meant to be familiar to *legions* of C++ developers that would migrate to Java.
- Class-based, object-oriented design.
 - Explicitly object-oriented
 - Cannot build procedural applications!
- Extensive class libraries included
 - Cross-platform!
 - Support for everything from threading to database access to building user interfaces.
 - This makes Java unique; many languages rely on third-party libraries.

Java Virtual Machine (JVM)

- Portability is achieved through virtualization
 - Java compiles to bytecode (IR).
 - Bytecode is executed by a Java virtual machine (JVM) on the target platform.
 - Interpreted bytecode is slower than native code BUT just-in-time compilation can give near-native performance.

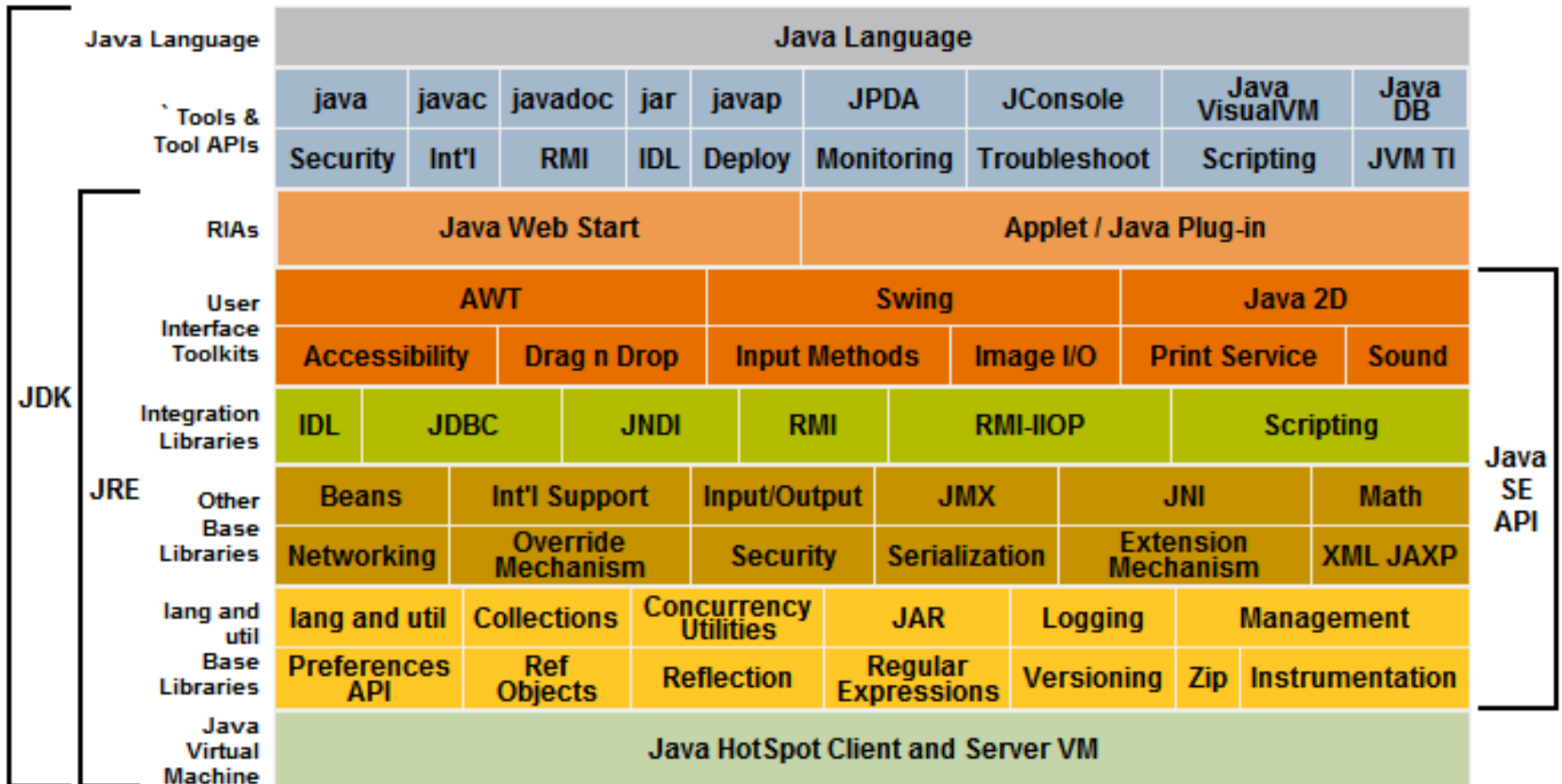


<http://viralpatel.net/blogs/java-virtual-machine-an-inside-story/>

- There are two main Java implementations
 - Oracle Java: <https://docs.oracle.com/javase/8/>
 - Open JDK: FOSS implementation for Linux.
- JRE: standalone JVM installation (runtime).
- JDK: JRE plus development tools and libraries.
 - This gives you command-line tools (javac compiler and java runtime).
- Third-party support is excellent
 - Editor support in VIM, Sublime, etc.
 - IDEs like IntelliJ, Eclipse.

Java Platform (JDK)

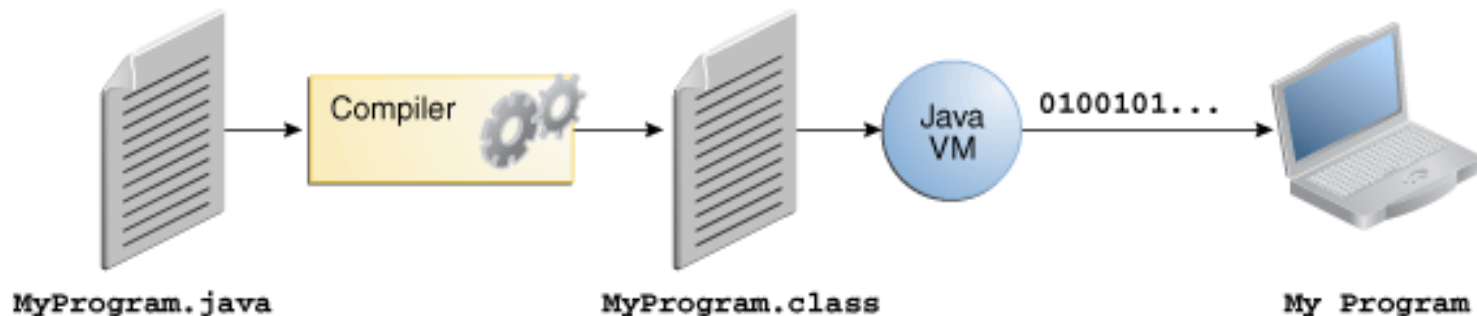
- Includes tools, and libraries - everything from threading, to database access, to UI toolkits – all cross platform and portable.



Description of Java Conceptual Diagram

Building Applications

- Source code is written in text files ending with the `.java` extension (one class per source file).
- Source files are compiled into `.class` files (bytecode) by the `javac` compiler.
- Class files (`.class`) can be executed on any platform with an appropriate JVM.
- Often applications include many class files, which we bundle into a JAR (`.jar`) file (basically a zip file with metadata).



To compile on the command line:

```
$ javac CountArgsApp.java
```

To run compiled app:

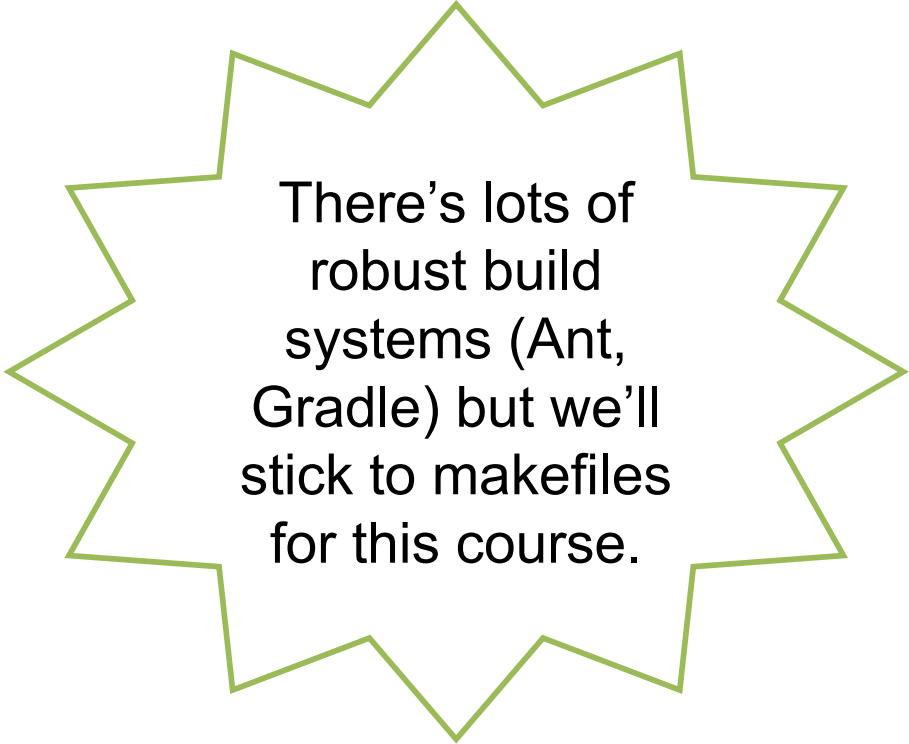
```
$ java CountArgsApp one two three  
You provided 3 args.
```

This works when you have a single source file, but starts to fall apart when you have multiple files and dependencies (files that depend on one another).

Example Makefile

```
main: ← target
    javac TestWindow.java ← command
run:
    java TestWindow
clean:
    rm *.class
```

```
Syntax: make [target]
$ make
javac TestWindow.java
$ make run
java TestWindow
```



There's lots of robust build systems (Ant, Gradle) but we'll stick to makefiles for this course.

The Java Programming Language

Language features

Classes and objects

Libraries

Program structure

- Java uses the same syntax as C++ in most situations, so the style will be familiar.
 - Variables
 - Standard naming conventions apply
 - Operators
 - Equality: `==` `!=`
 - Assignment: `=` `+=` `-=` and so on.
 - Logical: `not (!)` and `(&&)` or `(||)` ternary(`? :`)
 - Structure
 - `{ }` to nest blocks of code
 - Control flow: `if... else`, `while`, `break/continue`.

Java Simple Types: see C++

Type	Size	Format	Range
boolean	≤1 byte	—	false, true
byte	1 byte	signed integer	±127
short	2 bytes	signed integer	±32,767
char	2 bytes	unsigned integer	0 to +65,535
int	4 bytes	signed integer	±2,147,483,647
long	8 bytes	signed integer	... Umm, really big
float	4 bytes	floating point	±10 ³⁸
double	8 bytes	floating point	±10 ³⁰⁸

Primitive types also have corresponding Object types (e.g. Boolean class for booleans, Integer for integers and so on.)

Familiar OO features still apply to Java:

Feature	Description
Classes, objects	Types, instances of those types
Dynamic dispatch	Objects determine type at runtime, instead of being static typed.
Encapsulation	Data hiding
Composition	Nesting objects
Inheritance	Hierarchy of is-a class relationships
Polymorphism	Calling code is agnostic in terms of whether an object is a derived or base class.

Java Class Structure

Variables

Class variables	Shared among all objects.	static
Instance variables	Belong to an object instance.	
Member variables	All variables belonging to an object.	

Methods

Class methods	Shared among all objects.	static
Instance methods	Belong to an object instance.	

Structure of a Program

Bicycle.java

```
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());

    }
}
```

Instantiating Objects

- In Java,
 - Primitive types are allocated on the stack, passed by value.
 - Objects are allocated on the heap, passed by reference
 - Technically, value of address passed on the stack, but behaves like pass-by-reference.

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

Both refer to the same memory on the heap

- Practically, this means that you don't need to worry about pointer semantics in parameter passing.

Pointer Gotchas!

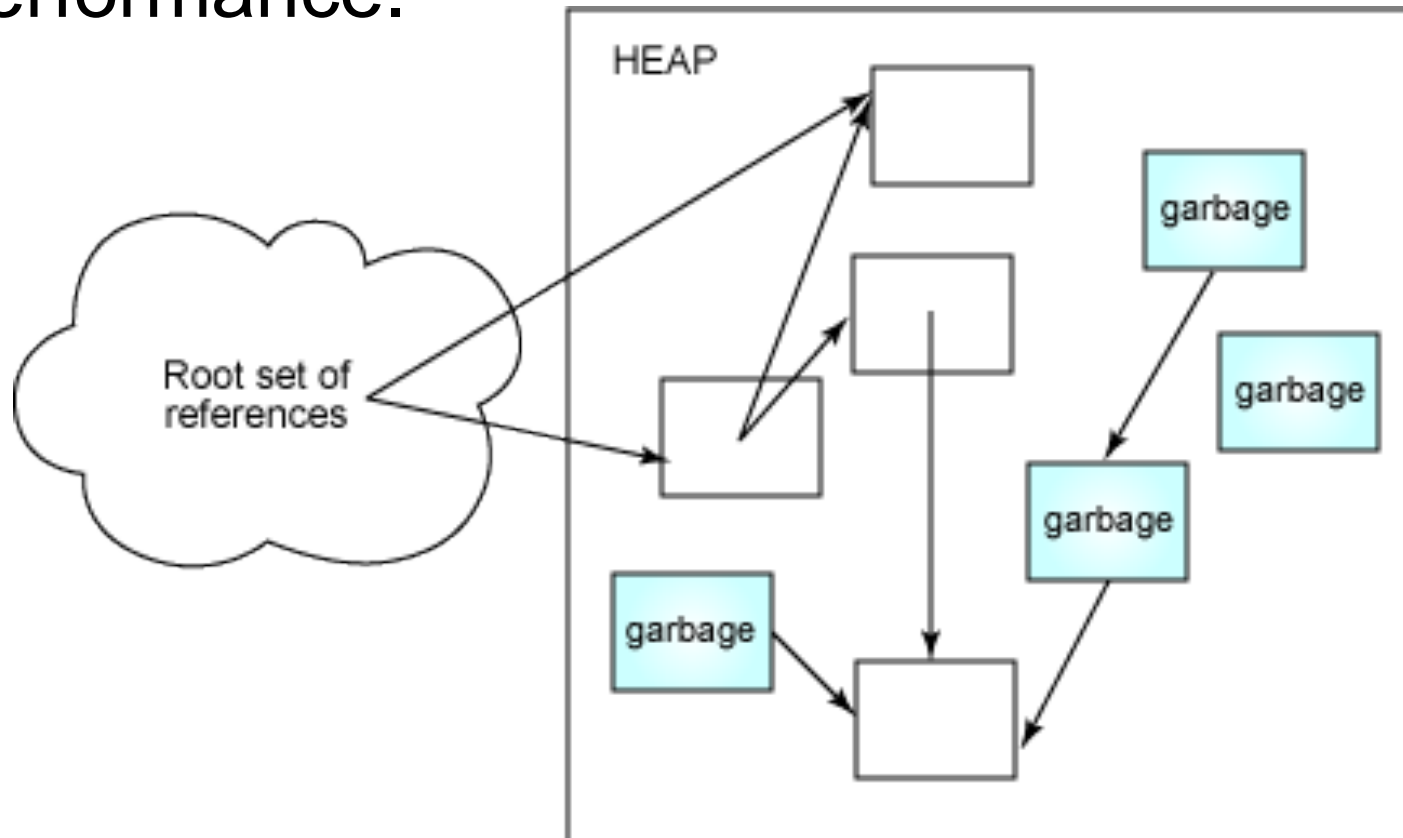
- Arrays are objects, so:

```
Bicycle[] bikeStore = new Bicycle[10];  
// bikeStore[0].changeSpeed(25); //!!! Null pointer  
for (int i=0; i<10; i++) {  
    bikeStore[i] = new Bicycle();  
}  
bikeStore[0].changeSpeed(25); // No problem
```

- First line allocates an array on the heap with space for 10 Bicycle pointers (40 bytes), but all Bicycle pointers are still null
- for loop allocates space for Bicycle to each entry in bikeStore array

Garbage Collection (GC)

- In Java, there's no need to free memory
 - Garbage collection runs periodically and frees up memory that's not in use.
 - JVM attempts to do this without impacting performance.



<http://www.ibm.com/developerworks/library/j-jtp10283/>

Structure of a Program

Bicycle.java

No destructor required
for this class!

```
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());
    }
}
```

Composition: Inner Classes

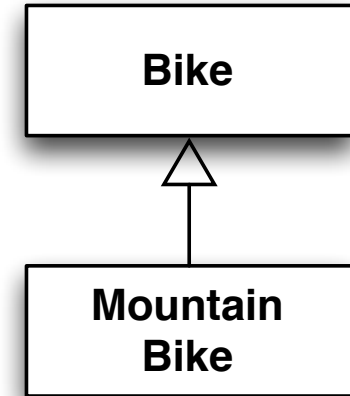
Classes can be nested as inner classes. Watch scope!

```
class ShadowTest {  
    public int x = 0;  
  
    // inner class  
    class FirstLevel {  
        public int x = 1;    // this is a different x!  
  
        void methodInFirstLevel(int x) {  
            System.out.println("x = " + x);  
            System.out.println("this.x = " + this.x);  
            System.out.println("ShadowTest.this.x = "  
                + ShadowTest.this.x);  
        }  
    }  
}  
  
public static void main(String... args) {  
    ShadowTest st = new ShadowTest();  
    ShadowTest.FirstLevel fl = st.new FirstLevel();  
    fl.methodInFirstLevel(23);  
}  
}
```

x = 23
this.x = 1
ShadowTest.this.x = 0

Inheritance

- Inheritance: increasing code reusability by allowing a class to inherit some of its behavior from a base class (“is a” relationship).
 - Classes inherit common attributes and behavior from base classes.
 - e.g. “Mountain Bike” is-a “Bike”.
 - Common: speed, gear
 - Unique: engine
 - Use ***extends*** keyword.



Subtype Polymorphism

```
public class Animals1 {  
  
    // inner classes  
    // base class  
    abstract class Animal {  
        abstract String talk();  
    }  
  
    class Cat extends Animal {  
        String talk() { return "Meow!"; }  
    }  
  
    class Dog extends Animal {  
        String talk() { return "Woof!"; }  
    }  
  
    // container class methods  
    Animals1() {  
        speak(new Cat());  
        speak(new Dog());  
    }  
  
    void speak(Animal a) {  
        System.out.println( a.talk() );  
    }  
}
```

Animals1.java

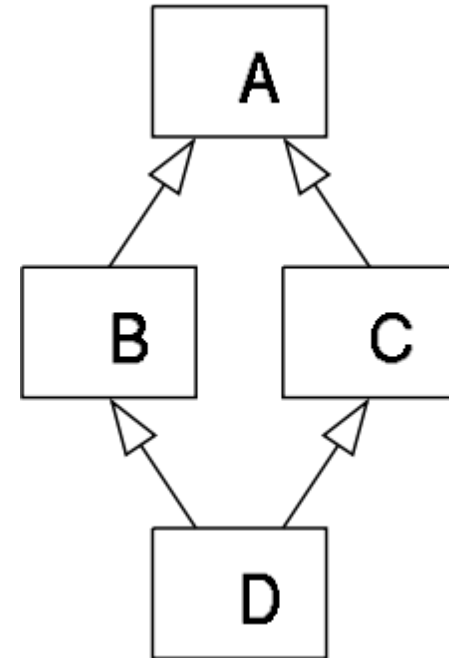
The *Animal* class is abstract, and cannot be instantiated.

It's *talk()* method is abstract, so derived classes must override it.

” Meow! “
“ Woof! “

Single Inheritance

- Java only supports single inheritance!
 - In practice, this simplifies the language.
 - See the “Diamond Problem” for one example of multiple inheritance being a hard problem.
 - Solutions to this problem vary by language; Java just doesn’t let you do it.
- It’s *very* common to derive an existing Java Platform class and override behavior.
- All classes have Object as their ultimate base class (implicit).



Interfaces

- An interface represents a set of methods that must be implemented by a class (“contract”).
- Similar to pure abstract classes/methods.
 - Can’t be instantiated
 - Class implementing the interface must implement all methods in the interface.
 - Use ***implements*** keyword.
- In Java,
 - *extend* a class to derive functionality from it
 - *implement* an interface when you want to enforce a specific API.

Interfaces Example

- We can replace the abstract class/method with an interface.
 - Polymorphism still applies to interfaces.

```
// interface
interface Pet {
    String talk();
}

// inner class
class Cat implements Pet {
    public String talk() { return "Meow!"; }
}

class Dog implements Pet {
    public String talk() { return "Woof!"; }
}
```

Animals2.java

```
void speak(Pet a) {
    System.out.println( a.talk() );
}
```

Note that we're treating the interface, Pet, as a type

- You can (and will often) mix approaches.
 - e.g. Drivable interface could be applied to any type of drivable vehicle, including our Bike class hierarchy.

```
interface Drivable {  
    public void accelerate();  
    public void brake();  
    public int getSpeed();  
}
```

```
// implement interface only  
class Car implements Drivable {}  
class Train implements Drivable {}
```

```
// derive from base class, and implement interface  
class Motorcycle extends Bike implements Drivable {}
```

Extended Example

Bikes1.java – classes
Bikes2.java - interfaces

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

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

Recommended Resources

- Required
 - Java SE 8 JDK :
<http://www.oracle.com/technetwork/java/javase/>
- Reference
 - Java 8 SE Platform SDK Documentation:
<https://docs.oracle.com/javase/8/docs/api/overview-summary.html>
 - Java 8 Tutorials:
<http://docs.oracle.com/javase/tutorial/java/index.html>
- IDE
 - IntelliJ (Jetbrains Student Licenses):
<https://www.jetbrains.com/student/>