

# Introduction to Java Programming

Rust Belt Robotics - 424

# Intro To Programming

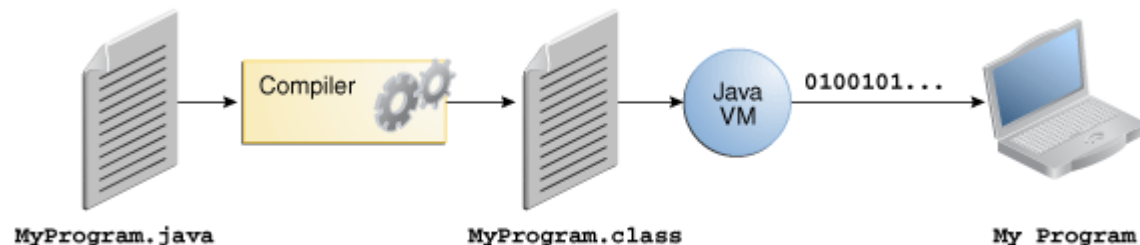
- What is Programming all about?
  - Solving complex problems by breaking them down into smaller more manageable chunks
  - Connect these smaller solutions together to solve the bigger problem
- Algorithm
  - Steps for solving a problem
  - Write out the steps (in English or “pseudocode”)
  - Convert these steps into code (write the implementation)

# What is Java?

- A high-level programming language
- **Object oriented**
- High performance
- “Write once, run everywhere”
  - Same app can run on Windows, Mac, Linux, etc. without recompiling
- Used extensively in real world applications
  - Enterprise (business)
  - Embedded devices (Smart devices, cars, etc.)
  - Web based applications

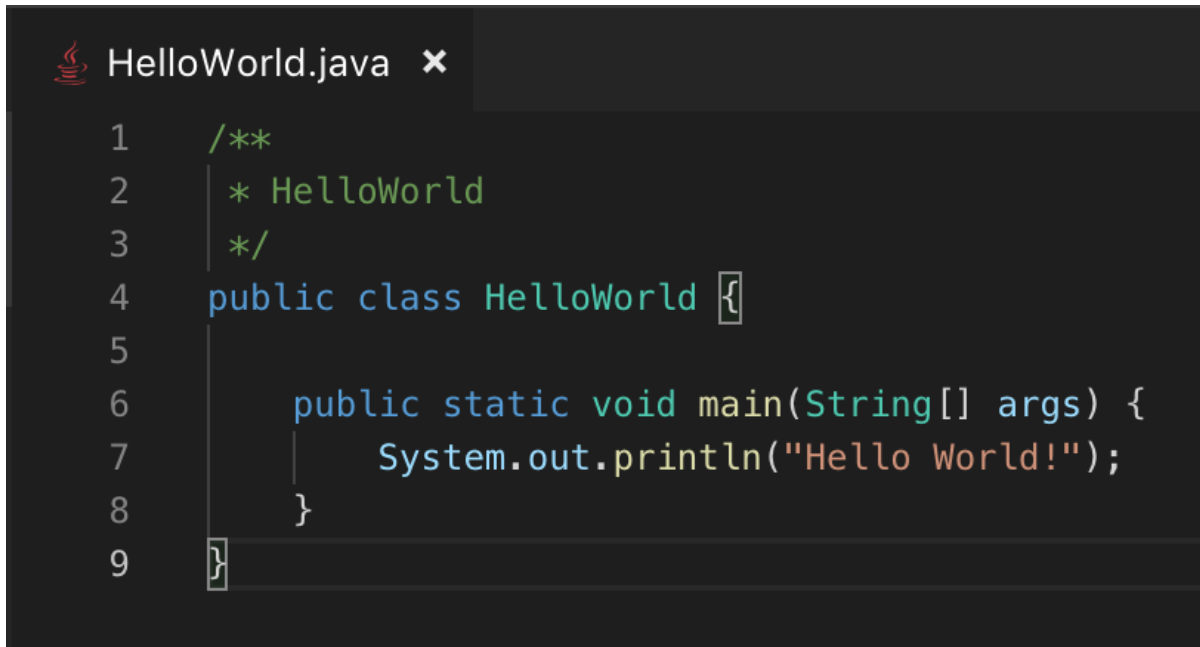
# What is a Java Application?

- Collection of source code files (text files with .java extension)
- The Java Compiler (javac) is run to convert source files into bytecode (.class files)
- The Java Virtual Machine (JVM = “java” command) is run
  - Executes your code
  - Compiles frequently run code to native code (faster)
  - Performs other (runtime) optimizations to make your app run faster



# What does a Java App look like?

- One or more **Classes**
- A **main** method
  - This is the starting point into your application logic flow



```
1  /**
2   * HelloWorld
3   */
4  public class HelloWorld {
5
6      public static void main(String[] args) {
7          System.out.println("Hello World!");
8      }
9  }
```

# What is an Object?

- **State**

- Attributes
  - E.g. Every Dog has: name, color, breed, weight, height
- Modeled using class **fields** in Java

- **Behavior**

- Actions
  - E.g. Dogs can: bark, fetch, wag tail
- Change State
- Modeled using class **methods** in Java
  - Methods are sometimes also referred to as *functions*
  - Method format:
    - `<access modifier> <return type> <methodName>(<parameters>) <throws clause> {`  
    `<method body>`  
    `<return statement – optional for void return type>`  
    `}`

# What is a Class?

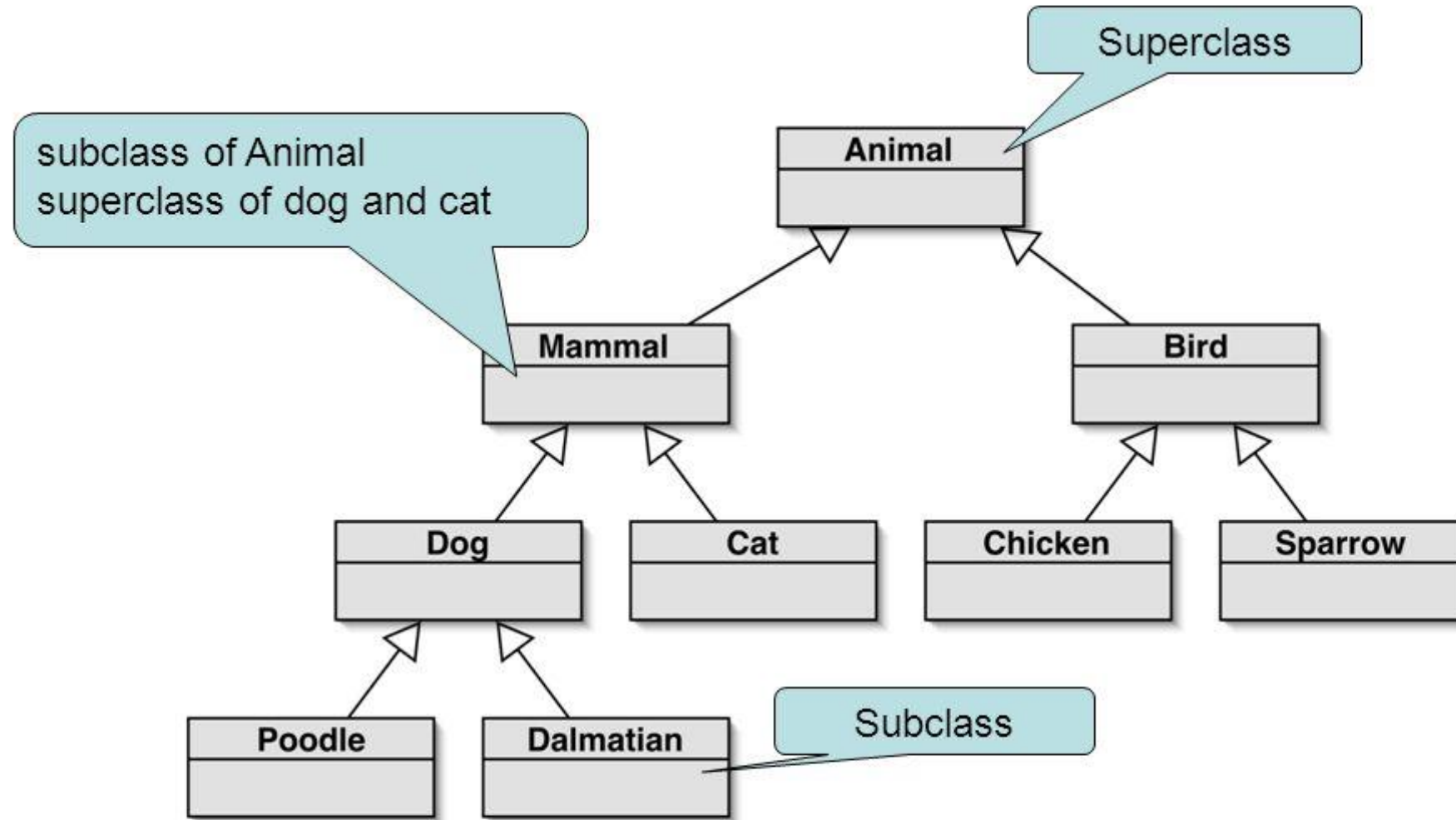
- A **Class** is used to model Objects
- The class is like a blueprint or factory for constructing object **instances**
  - Instances are created using the **new** keyword
    - `Dog myDog = new Dog("Chewy");`
    - `Dog yourDog = new Dog("Rex");`
- Each object instance is independent and occupies different memory
- Classes can be defined/nested within other Classes

# What is Inheritance?

- Different kinds of objects often have a certain amount in common with each other
- Object-oriented programming allows classes to **inherit** commonly used state and behavior from other classes using the **extends** keyword
- Each class is allowed to have one direct superclass (parent), and each superclass has the potential for an unlimited number of subclasses (children)
- This inheritance graph can be visualized as a tree structure with the base class at the root and children extending down to leaf nodes



# Inheritance



# What is an Interface?

- An **interface** is a group of related methods with empty bodies
- It defines the behavior (the “what”) for classes that **implement** (the “how”) the interface (that is, provide the method body)
- All methods are public by default (unless you specify otherwise)
- An interface can extend another interface
- An interface cannot be instantiated (have object instances created)
- An interface *may* define a **default** implementation of methods
  - This implementation will be used by all classes that implement the interface
  - The default implementation may be overridden (that is, redefined)
- Helps to provide abstraction and loose coupling between components
  - Code declares objects of the interface type rather than the concrete/implementing subclass
  - Allows you to more easily swap out the implementation being used

# What is an Abstract Class?

- Abstract classes can provide properties (variables) and methods that are common/shared among all subclasses
- Methods with the **abstract** keyword cannot have an implementation – it must be supplied by a subclass
  - Useful in cases that you know a behavior/method is common among subclasses, but that the implementations must be different
- Abstract classes cannot be instantiated
  - Only non-abstract subclasses of an abstract class can be instantiated

# What is a Package?

- A **package** is a namespace that organizes a set of related classes and interfaces
- Conceptually you can think of packages as being similar to different folders on your computer
- Before you reference a class or interface you must first **import** it to make it visible to the compiler  
E.g. `import java.util.List;`
- Everything in the `java.lang` package and the same package as your current `.java` source file is automatically visible/imported

# Primitive (non-object) Data Types

Data Type	Memory Size	Value Range	Default Value	Description
byte	8 bits (1 byte)	-128 to 127	0	Integers (whole numbers)
short	16 bits (2 bytes)	-32,768 to 32,767	0	Integers (whole numbers)
int	32 bits (4 bytes)	$-2^{31}$ to $2^{31}-1$	0	Integers (whole numbers)
long	64 bits (8 bytes)	$-2^{63}$ to $2^{63}-1$	0L	Integers (whole numbers)
float	32 bits (4 bytes)	<a href="#">See here</a>	0.0f	Decimals (imprecise!) IEEE 754 Floating Point
double	64 bits (8 bytes)	<a href="#">See here</a>	0.0d	Decimals (imprecise!) IEEE 754 Floating Point
boolean	~ 1 bit	true or false	false	Logic
char	16 bits (2 bytes)		'\u0000'	Single Unicode character, Eg. 'A' or 'B'
String	Variable		null	Sequence of characters representing text. Technically an Object and not a primitive type.

# Arrays

- An array is a container object that holds a **fixed** number of values of a **single type**
- The **length** of an array is established when the array is created
  - After creation, its length is fixed (it cannot be changed)
- Eg.
  - Declaration: `int[] myArray = new int[10];` //array can hold up to 10 items
  - Declaration and initialization: `int[] anArray = { 10, 20, 30 };`
- Values are accessed by index (starting with 0, not 1)
  - Eg. `anArray[1]` //this value is 20
  - Eg. `anArray.length` //3 (3 items: 10, 20, 30)

# Expressions, Statements

- An **expression** is a construct made up of variables, operators, and method invocations, that evaluates to a single value
- Statements are roughly equivalent to sentences in natural languages, but instead of ending with a period, a statement ends with a semicolon ( ; ). A **statement** forms a complete unit of execution.
  - Assignment expressions
    - `aValue = 8933.234;`
  - Any use of ++ (increment) or – (decrement)
    - `aValue++;`
  - Method invocations
    - `System.out.println("Hello World!");`
  - Object creation expressions
    - `Bicycle myBike = new Bicycle();`

# Blocks

- A **block** is a group of zero or more statements between balanced braces and can be used anywhere a single statement is allowed

```
if (condition) { // begin block 1
    System.out.println("Condition is true.");
} // end block one
else { // begin block 2
    System.out.println("Condition is false.");
} // end block 2
```



# Comments

- Single line comments are preceded by `//` (double forward slash)
- Multi-line comments are surrounded by `/* your code in here */`
- Comments serve multiple purposes
  - Code documentation
    - Generation of formatted API docs via [Javadoc](#)
      - Classes, methods, fields
      - Used by IDEs for auto-completion/doc popups
      - Doc pages can be published online to
    - Inline explanations, notes
      - E.g. TODO: reminders, algorithm/flow explanations, etc.
  - Temporarily removing code from program execution path
- Don't underestimate the value comments add to your program!
  - Well written comments can help you or a fellow coder more easily understand code
    - Remembering exactly how something works weeks, months or years after it was written isn't always as easy as you might think!

# Type Casting

- Type casting is when you assign a value of one primitive data type to another type
- Widening (automatic)
  - Converting a smaller type to a larger size type
  - byte -> short -> char -> int -> long -> float -> double
- Narrowing (manual)
  - Converting a larger type to a smaller size type
  - double -> float -> long -> int -> char -> short -> byte
  - Precision will be lost in the conversion!

# Access Control

- Access level modifiers determine whether other classes can use a particular field or invoke a particular method
- There are two levels of access control:
  - At the top (class) level - public, or package-private (no explicit modifier)
  - At the member level - public, private, protected, or package-private (no explicit modifier)

	default	private	protected	public
same class	yes	yes	yes	yes
same package subclass	yes	no	yes	yes
same package non-subclass	yes	no	yes	yes
different package subclass	no	no	yes	yes
different package non-subclass	no	no	no	yes

# Enum Types

- An *enum type* is a special data type that enables for a variable to be a set of predefined constants
  - These values can be easily *enumerated* (counted over), hence the name: enum
- The variable must be equal to one of the values that have been predefined for it
- Because they are constants, the names of an enum type's fields are in uppercase letters
- The enum declaration defines a class (called an enum type)
- The enum class body can include methods and other fields

# High Level Concepts

- API
  - “Application Programming **Interface**”
  - Contract on how systems or modules communicate
  - Describes **what** a method/function does, but **not how**
    - Inputs
    - Outputs
    - Expected conditions
- Library
  - A **re-usable** collection of code
  - Typically focused on a specific type of problem

# References – Study Materials

- <https://docs.oracle.com/javase/tutorial/java/nutsandbolts/index.html>
- <https://www.geeksforgeeks.org/java-cheat-sheet/>