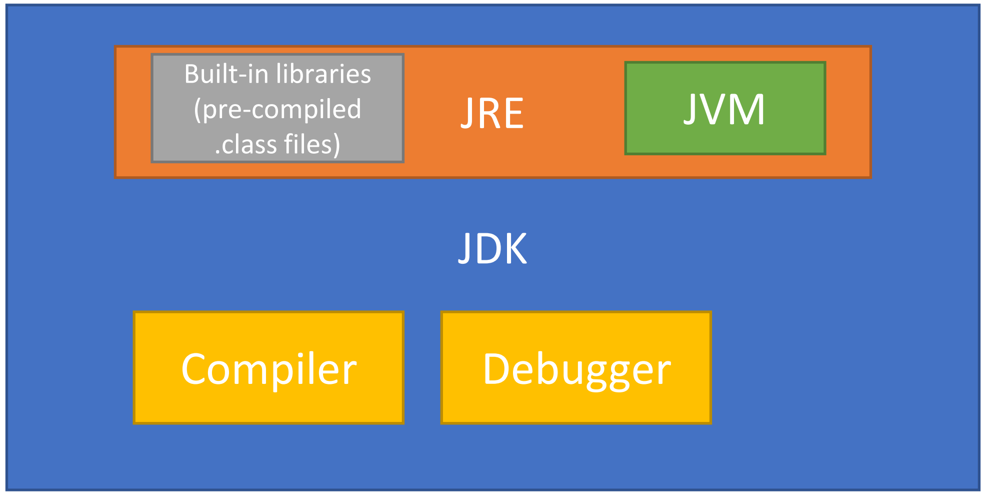
Software Setup

- JDK (Java Development Kit) 11
- IntelliJ IDE

JDK, JRE, JVM

- JDK
  - Java Development Kit
  - **Compiler**
  - Debugger
  - Other development tools
  - Contains a JRE
- JRE
  - Contains all of Java's built-in libraries
  - Contains a JVM
- JVM
  - Java Virtual Machine
  - Runs the program (bytecode)



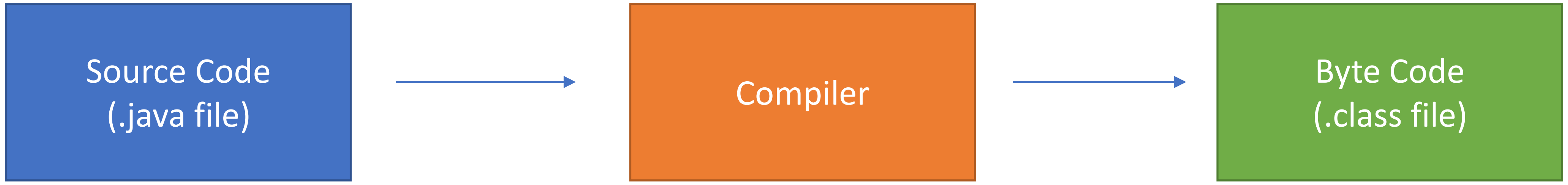
If you are a software developer, download and install a JDK

If you are not a software developer and just want to run a Java program that somebody else has created, download and install a JRE

JRE contains everything to run a Java program
JDK contains everything to develop and run a Java program

Source Code to Byte Code

- Compiler included with the JDK compiles our source code into byte code
    - Source Code: the code we as humans write
    - Byte code: a Java standard for representing a program in a lower-level representation (more optimized and efficient)
- When we run a Java program, the JVM included in the JRE executes the byte code



Variables in Java

- 2 types of variables
  - Reference variables
  - Primitive variables
- In Java, there is the idea of primitive datatypes and objects
  - A reference variable refers to the location in memory where an object exists
  - A **primitive variable** stores the primitive value

Primitive Datatypes

- Strings are not primitives in Java, they are **objects**!
- byte: -128 to 127
- short: -32768 to 32767
- **char**: ‘a’ (single character)
- **int**: -2,147,483,648 to 2,147,483,647
- long: -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
- float: stores decimal numbers sufficient for 6 to 7 decimal digits
- **double**: stores decimal numbers sufficient for 15 decimal digits
- **boolean**: true or false

Creating a new Datatype

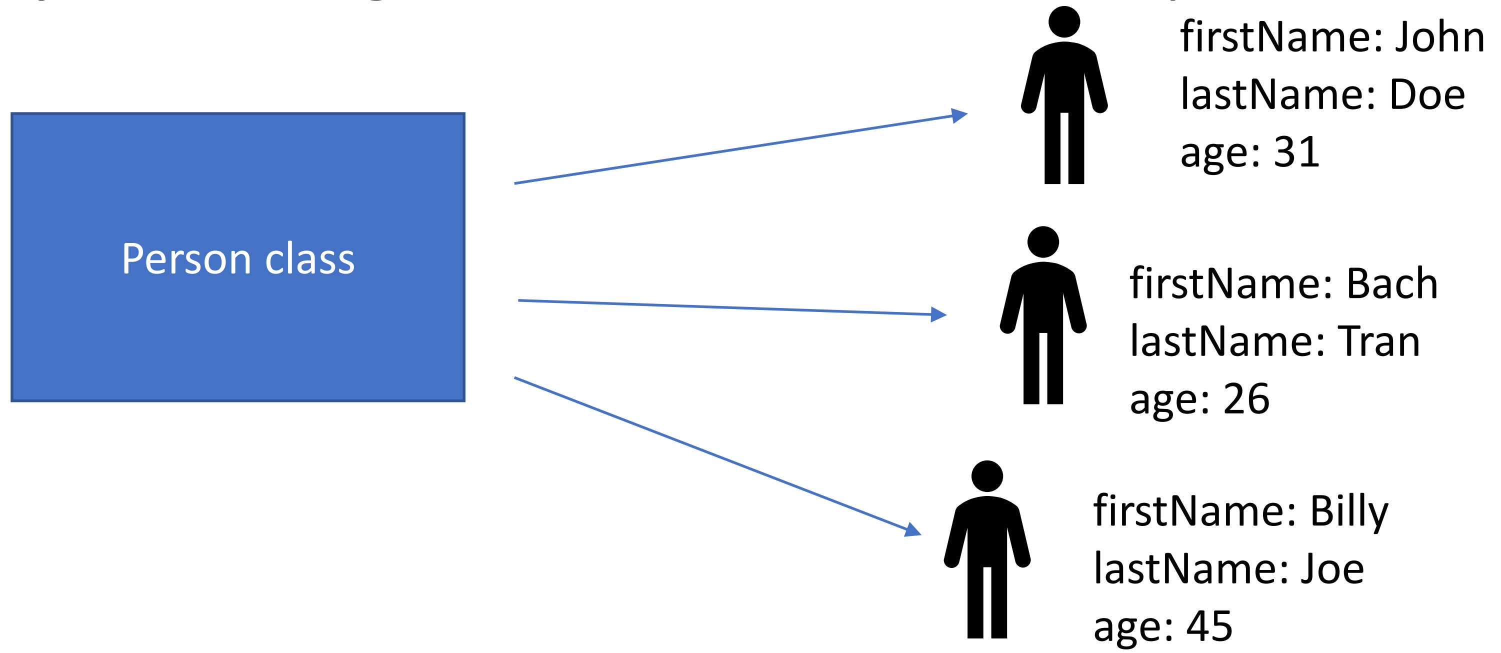
- You cannot create your own primitive datatypes
- However, you can create your own reference datatypes
  - Creating a class or interface establishes a new datatype with the name of that class
    - However, this is not the primary purpose of a class, the main purpose of a class is to create a “blueprint” for constructing objects
  - ex. Create a class called Person will create a datatype called Person as well

Classes

- Classes act as “blueprints” for creating objects
- Java is an object oriented programming language and therefore revolves around classes
- A class contains several different components
    - Fields / properties: variables that are scoped to individual objects
    - Constructor(s): used to create an object and pre-populate the object with certain properties
    - Methods: functions to define behaviors of those objects

Classes v. Objects

- Classes and objects are different from each other, but they are certainly related
- Classes = blueprint for creating objects
- Object = a single manifestation of a blueprint



Variable Scopes

- static scope (global scope): a variable that exists in a class itself
- instance scope: a variable that exists in a particular object
- method scope: variable that exists while a given function is executing
- block scope: a variable that exists inside of a code block such as a for loop, if statement, etc.

If, else-if, else, switch, for loops, while loops, etc. statements

- Look exactly the same as in JavaScript
- JavaScript v. Java
  - Java has mandatory semicolons at the end of each statement
    - JavaScript does not
  - Not related at all
  - Keep in mind the differences in the structure of each programming language
    - Java code always exists in classes
    - in JavaScript, you don’t need to put code into classes

4 Pillars of OOP

- A PIE
- Abstraction (4)
- Polymorphism (3)
- Inheritance (2)
- Encapsulation (1)

Access modifiers

- Access modifiers are applied to fields and methods
- 4 different types of access modifiers (most accessible to least accessible)
    - public: a field or method that can be accessed from anywhere
    - protected: a field or method that can be accessed by other classes that exist inside of the same package OR a class that is a subclass of another class (no matter what package the subclass is in)
    - default: a field or method that can be accessed by other classes that exist inside of the same package
        - default is the absence of a keyword. There is a default keyword in Java, but it is used for a different purpose, not for access modifiers
    - private: a field or method accessible only from within the same class

Encapsulation

- Encapsulation refers to the bundling of data (fields) and behaviors (methods) that operate on that data within a single unit called a class
- Encapsulation restricts direct access to an object's data and ensures that the object's internal state can only be modified through specific methods. “Data hiding”
- Encapsulation is achieved using **access modifiers** and **getter and setter** methods
  - getters/setters: to access or modify private fields, getters (accessor) and setters (mutator) are used. These are typically marked a public to allow controlled access to the private fields
  - **Preventing direct data access prevents arbitrary modification of the state of an object**
    - ~~**p.age = 1000;**~~
    - **p.setAge(1000); // Refuse to set age to 1000 since it’s outside of a reasonable range**
- Common misconception is that access modifiers are about security
  - They are simply there to make sure that other developers using your code are not using it in ways that weren’t intended in the original design of that code

Inheritance

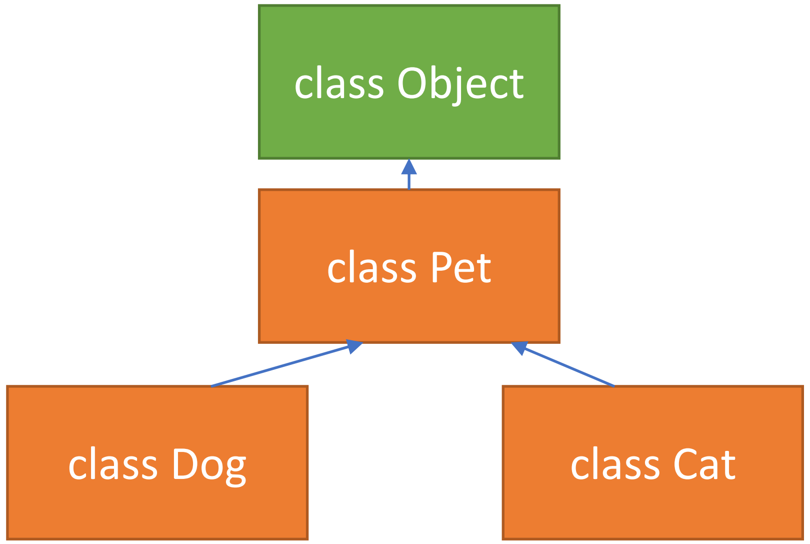
- Where one class inherits the methods and fields defined in another class
- It is used to remove redundancies in your code (DRY “don’t repeat yourself” principle)
- Inheritance forms an IS-A relationship
- Terms
  - Parent class / superclass
  - Child class / subclass: inherits the fields and methods of the parent class

Polymorphism

- Means “many forms”
- Achieved through method overloading and method overriding
    - Method overloading: where 2 or more methods in the **SAME** class have the **same name** but a **different number and/or type of parameters** to distinguish each method
    - Method overriding: Where a method in a child class “overrides” the implementation of a method with **same name** and **same parameters** in the parent class

Object Class

- If a class does not extend another class, it will implicitly extend the "Object class"
  - All classes in Java are children of the Object class either directly or indirectly
  - Dog and Cat are indirect children, Pet is a direct child



Object Class Methods

- The object class provides several methods that are available to every object in Java because of the previously explained inheritance structure
  - public String toString()
    - Returns a string representation of an object
    - Is often overridden in custom classes to provide a meaningful representation of the object’s state
  - public boolean equals(Object obj)
    - Checks if the current object is equal to the specified object (obj)
    - By default, compares the memory addresses of the two objects
    - However, is usually overridden to provide a more meaningful comparison based on the objects’ attributes
  - public int hashCode()
    - Return the hash code value of an object
    - Used in data structures such as HashSet and HashMap to store and retrieve objects more efficiently
    - When overriding the equals method, it is important to also override the hashCode method to maintain the “contract” between the two methods
      - “contract”: If two objects are equal, they MUST have the same hashCode. But, if two objects have the hashCode, it does not mean they must be equal

Abstraction

- Exposing only the essential features and allows us to focus on the “what” rather than the “how”
  - Hiding the complexity of implementation details
- Abstraction is achieved through the use of abstract methods, which can either be defined in **abstract classes** or **interfaces**
  - Abstract class: a class that cannot be instantiated and must be used as a superclass for other classes (since they can’t be instantiated)
    - An abstract class can have both abstract and non-abstract methods
  - Interface: a collection of **abstract methods** that specifies a contract that implementing classes must follow
    - In Java 8, the default keyword was introduced which allows for defining methods that have an implementation in an interface
- In Java, a class can only extend 1 other class, but a class can implement as many interfaces as it wants
- Abstract method: a method without any implementation (no code)
  - A non-abstract class that extends an abstract class or implements interfaces, must override and provide an implementation for all abstract methods that it inherits