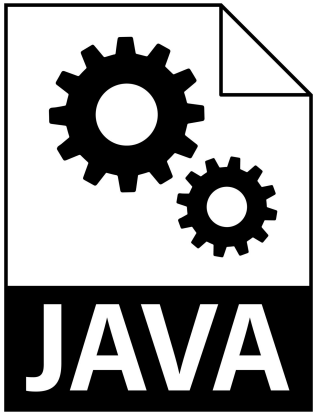
CISC 3115: Introduction to Programming Using Java



Summer 2025 Prep Workshop

Itinerary

01 Review of Java Fundamentals
- The Structure of a Java Program
- Basic Syntax Rules

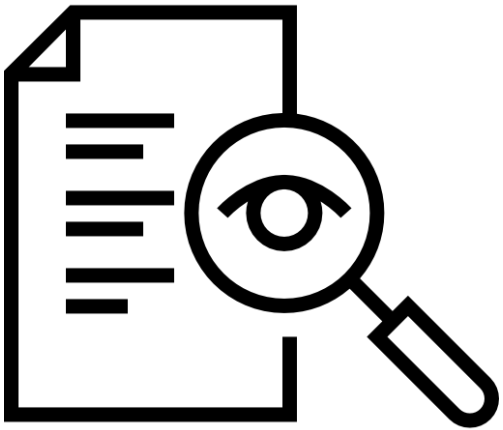
02 What is Object-Oriented Programming (OOP)?

03 Classes and Objects

04 Inheritance, Abstract Classes, and Interfaces

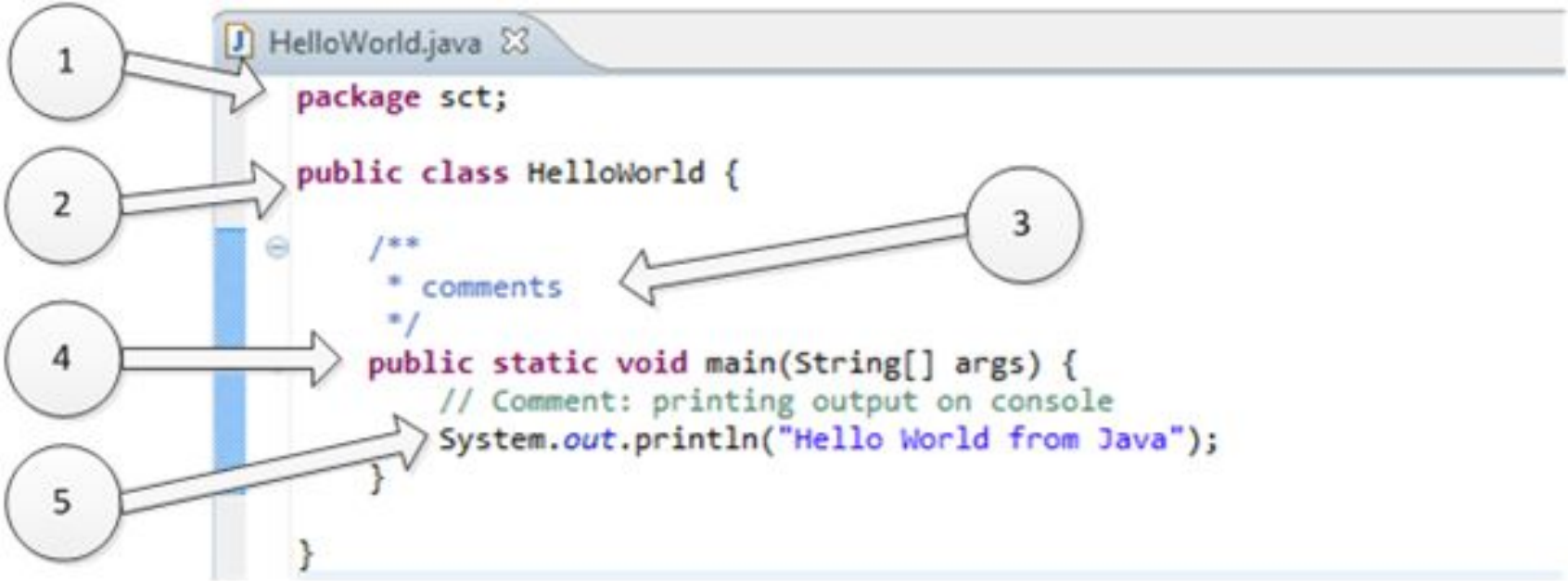
05 Overview and Next Steps

01
Review of Java Fundamentals



The Structure of A Java Program

- As you already know, the structure of a basic Java program is as follows:



The Structure of A Java Program

- First, is the **package declaration statement**:
  - It **defines a namespace in which classes are stored**.
  - It is used to **organize the classes based on functionality**.
  - If you omit the package statement, the class names are put into the default package, which has no name.
  - Package statement **cannot appear anywhere in the program.**
  - It **must be the first line of your program** or you can omit it.

The Structure of A Java Program

- Second, is the **class declaration**:
  - This line has various aspects of java programming:
    - **public**: This is **access modifier keyword which tells compiler access to class.**
    - Various values of access modifiers can be public, protected, private or default (no value).
    - **class**: This keyword is **used to declare a class.**
    - The name of class (HelloWorld) followed by this keyword.

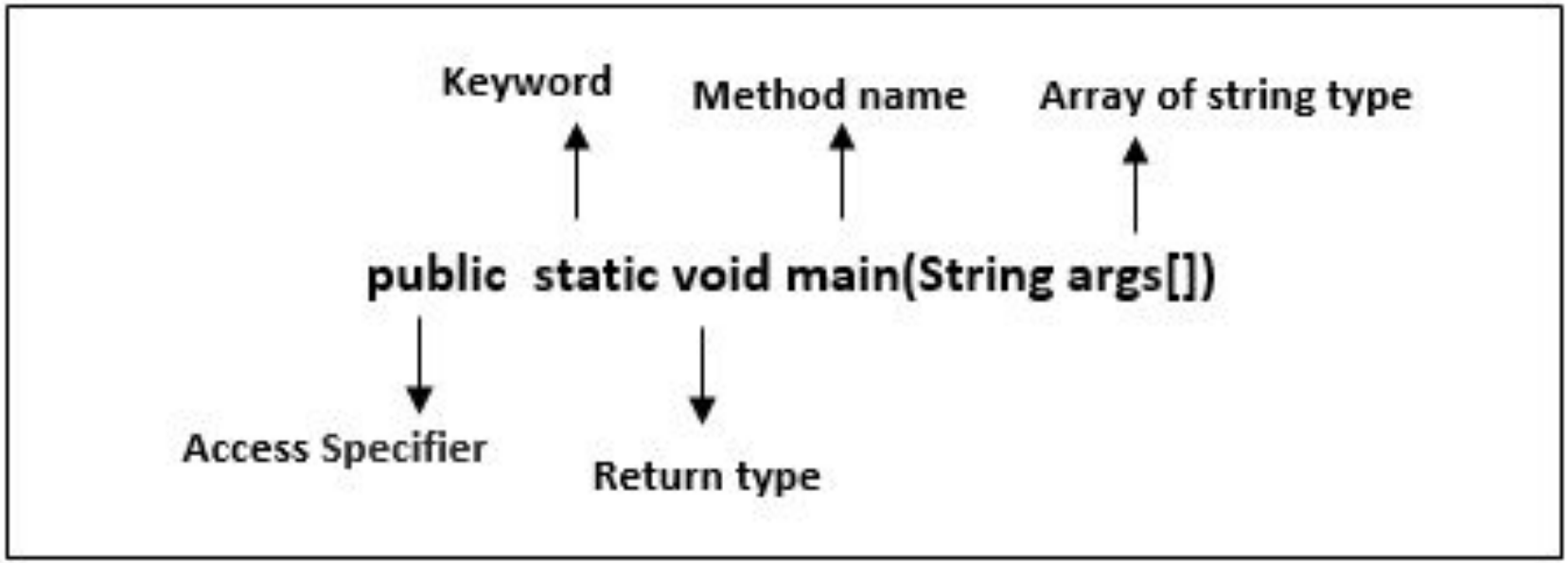
The Structure of A Java Program

- Third, is **the comment section**:
  - There are three types of Java comments:
    - **Single-Line Comments**: Start with // and extend to the end of the line.
    - **Multi-Line Comments**: Start with /* and end with */.
      - They can span multiple lines.
    - **Documentation Comments**: Start with /** and end with */.
      - These are used to create formal documentation using Javadoc.

The Structure of A Java Program

- Fourth, is **the main method declaration**:
  - It is **essential for all executable Java programs**, because the execution of all Java programs starts from the main() method (regardless of other methods).
  - In other words, it is an entry point of the class.
  - It must be inside the class.
  - It is made up of six distinct parts: t**he access modifier, the static keyword, the return type, the name/signature, the method parameters, and the method body**

The Structure of A Java Program



The Structure of A Java Program

- **public**: is an access specifier
  - We should use a public keyword before the main() method so that JVM can identify the execution point of the program.
  - If we use private, protected, and default before the main() method, it will not be visible to JVM
- **static**: You can make a method static by using the keyword static
  - We should call the main() method without creating an object.
  - Static methods are the method which invokes without creating the objects, so we do not need any object to call the main() method.

The Structure of A Java Program

- **void**: this is the return type
  - In Java, every method has the return type.
  - The void keyword acknowledges the compiler that main() method does not return any value.
- **main()**: It is a default signature which is predefined in the JVM.
  - It is called by JVM to execute a program line by line and end the execution after completion of this method.
- **String [] args**: The main() method also accepts some data from the user.
  - It accepts a group of strings, which is called a string array.
  - It is used to hold the command line arguments in the form of string values.

The Structure of A Java Program

- Lastly, there is **a print statement** inside of the body of the main method.

- This particular print statement is made up of four parts:
  - **System**: the name of Java utility class.
  - **out**: an object which belongs to System class.
  - **println**: A utility method name which is used to send any String to the console.
  - **“Hello World from Java”**: a String literal set as argument to println method.

Basic Syntax Rules

- About Java programs, it is very important to keep in mind the following points:

- **Case Sensitivity** - Java is case sensitive, which means identifier Hello and hello would have different meaning in Java.

- **Class Names** - For all class names the first letter should be in Upper Case. If several words are used to form a name of the class, each inner word's first letter should be in Uppercase.
  - Example: class MyJavaClass

Basic Syntax Rules

- **Method Names** - All method names should start with a Lowercase letter. If several words are used to form the name of the method, then each inner word's first letter should be in Uppercase.
  - Example: public void myMethodName()

- **Program File Name** - Name of the program file should exactly match the class name (Remember Java is case sensitive) and append '.java' to the end of the name (if the file name and the class name do not match, your program will not compile).
  - Example: If 'MyJavaProgram' is the class name, the file should be saved as 'MyJavaProgram.java'

Basic Syntax Rules

- All Java components require names. Names used for classes, variables, and methods are called **identifiers**.

- In Java, there are several points to remember about identifiers. They are as follows:
  - **All identifiers should begin with a letter (A to Z or a to z), currency character ($) or an underscore (_).**
  - **After the first character, identifiers can have any combination of characters.**
  - **A keyword cannot be used as an identifier.**
  - **Most importantly, identifiers are case sensitive.**
  - Examples of legal identifiers: age, $salary, _value, __1_value.
  - Examples of illegal identifiers: 123abc, -salary.

02

What is Object-Oriented Programming (OOP)?

What is Object-Oriented Programming (OOP)?

- Procedural programming (like C, COBOL, BASIC, etc.) which is about writing procedures or methods that perform operations on the data, while **object-oriented programming (OOP)** is a **fundamental programming paradigm based on the concept of "objects"**
- These objects can **contain data in the form of fields (often known as attributes or properties) and code in the form of procedures (often known as methods).**
- The core concept of the object-oriented approach is **to break complex problems into smaller objects.**

What is Object-Oriented Programming (OOP)?

- Object-oriented programming has several advantages over procedural programming:
  - **OOP is faster and easier to execute**
  - **OOP provides a clear structure for the programs**
  - **OOP helps to keep the Java code DRY "Don't Repeat Yourself", and makes the code easier to maintain, modify and debug**
  - **OOP makes it possible to create full reusable applications with less code and shorter development time**
- Tip: You should extract out the codes that are common for the application, and place them at a single place and reuse them instead of repeating it

03

Classes and Objects

Classes and Objects

- You should be partially familiar with the concept of a Java Class from your previous class, but what exactly is a class?

- **A class is defined as a collection of objects.**

- You can also think of a class as **a blueprint from which you can create an individual object**.

- For example, Student is a class while a particular student named Tom is an object.

- To create a class, **we use the keyword class**.

Classes and Objects

- Properties of Java Classes:
  - It is not a real-world entity. It is just **a template or blueprint or prototype from which objects are created**.
  - It does not occupy memory.
  - It is a group of variables of different data types and a group of methods.
  - It can contain:
    - **Data member(s)**
    - **Method(s)**
    - **Constructor(s)**
    - **Nested Class(s)**
    - **Interface(s)**

Classes and Objects

- Properties of Java Classes:
  - It is not a real-world entity. It is just **a template or blueprint or prototype from which objects are created**.
  - It does not occupy memory.
  - It is a group of variables of different data types and a group of methods.
  - It can contain:
    - **Data member(s)**
    - **Method(s)**
    - **Constructor(s)**
    - **Nested Class(s)**
    - **Interface(s)**

Classes and Objects

- Most classes you write will have the keyword public before them though it is not required.
- Let's create a class called **Person**.
- Classes are almost always named with capitalized names though this is a matter of style, not a rule of the language.
- Here is the basic skeleton of a Person class:

```
public class Person
{
    // define class here - also called the "body" of the class

}
```

Classes and Objects

- You can create instances of the Person class with the keyword **new** as in **new Person()** and you can declare variables that can **hold a reference to a Person object with Person variableName.**
- Or put it altogether to declare some variables and initialize each one with a reference to a new Person as shown here:

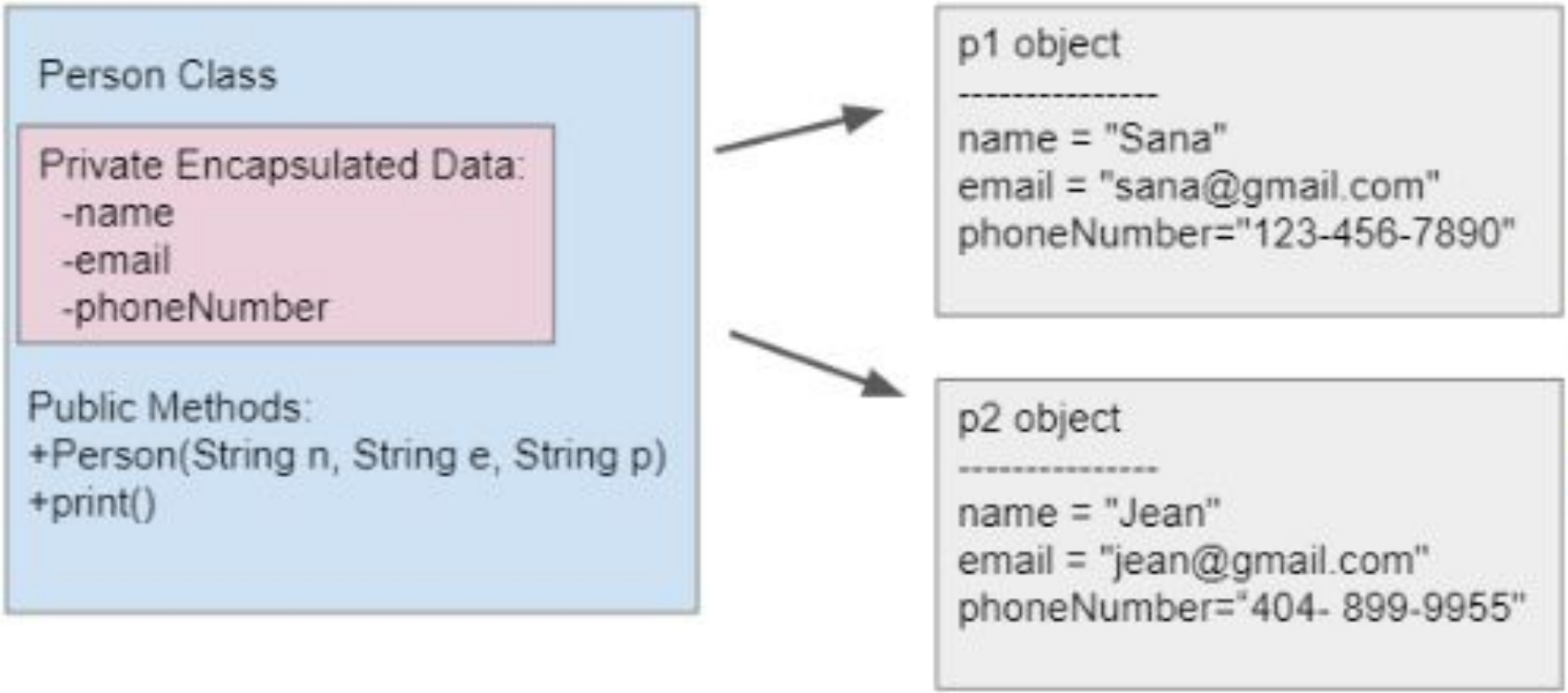
```
Person ada = new Person();
Person charles = new Person();
```

- So what makes up the body of the class?
- Remember that **objects have both attributes and behaviors.**
- These correspond to **instance variables and methods** in the class definition.

Classes and Objects

- The first things we define in a class are usually the **instance variables**.
  - They are called that because **each instance of the class (each object) has its own set of variables that aren't shared with other instances.**
- The next thing we define in a class is usually its **constructors**.
  - A constructor's job is to **initialize the instance variables when the object is created**. Usually that will mean they need to take arguments.
- Lastly, the **methods** of the class **define the behaviors of the objects of that class** and share access to the object's instance variables and when a method is called on an object it uses the instance variables for that object.

Classes and Objects



Classes and Objects

- As we've said, instance variables hold the data for an object. They record what an object needs to know to play its role in the program.
- They are also sometimes called **attributes, fields, or properties.**(Think of private as like your diary, only you should have direct access to it)
- Similarly, in Java **a private instance variable can only be accessed by code in the class that declares the variable.**
- Note: **Instance variables are declared right after the class declaration.**
- They usually start with private then the type of the variable and then a name for the variable. Private means only the code in this class has access to it.

Classes and Objects

- The Person class declares **3 private instance variables**:

```
// instance variables
private String name;
private String email;
private String phoneNumber;
```

- Once we have created a class like Person, we can create many instances (objects) of the class.
- Each object will have their own copies of the same instance variables but with possibly different values in them

Classes and Objects

- In the source code for a class, **constructors are usually written after the instance variables and before any methods.**
- The signature of a constructor is similar to the signature of a method **except there is no return type, not even void, and instead of a method name, the name of the constructor is the same as the name of the class.**
- The constructors you write will **almost always be marked public.**
- Like methods, constructors also **have a parameter list specified in parenthesis that declare the variables that will be used to hold the arguments passed when the constructor is called.**

Classes and Objects

- The easiest way to write a constructor is to not write one.
- If you do not write a constructor your class **will automatically get what is called the default no-argument constructor**.
- **This constructor will initialize all your instance variables to the default value for their type: 0 for int and double, false for boolean, and null for all reference types**.
- If those default values are sufficient to put your object into a valid state you may not need to write a constructor at all.

Classes and Objects

- Usually, however, if you are writing a class that has instance variables, you need to initialize your instance values to some other values.
- In that case you probably need to write a constructor that takes arguments and uses them to initialize your instance variables.
- For example, look at the constructor from the Person class:

```
public Person(String initName, String initEmail, String initPhone)
{
    name = initName;
    email = initEmail;
    phoneNumber = initPhone;
}
```

Classes and Objects

- This constructor ensures that all three of the instance variables (name, email, and phoneNumber) in Person are initialized to the values provided by whatever code called the constructor.
- For example, in the constructor call **new Person("Pat", "pat@gmail.com", "123-456-7890")**, the argument **“Pat” is passed into the parameter variable initName**, which the constructor then assigns to the instance variable name.
- One important note: **if you do write a constructor, Java will not generate the default constructor for you.**
- This is a good thing because it lets you **make sure that instances of your class are always properly initialized**. With this constructor in place, for instance, there’s no way to construct a Person object without providing the three required String values.

Classes and Objects

- Now to **methods** which **define what we can actually do with an object.**
- The most important methods in a class are the **public methods** since they can be accessed from outside the class.
- You may also write private methods that are not accessible outside of the class and therefore can only be used by other methods inside the same class.
- As you've probably figured out, the public and private keywords determine the external access and visibility of classes, instance variables, constructors, and methods.

Classes and Objects

- The Person class has a void print method that takes no parameters and prints out all the data stored for a person object.
- As we've discussed, the method can access and use the instance variables defined in the class: name, email, and phoneNumber but will get the values specific to the object we called print on.

```
public void print()
{
  System.out.println("Name: " + name);
  System.out.println("Email: " + email);
  System.out.println("Phone Number: " + phoneNumber);
}
```

```java
public class Person {
    //instance variables
    private String name;
    private String email;
    private String phoneNumber;

    //constructor
    public Person (String n, String e, String pN) {
        name = n;
        email = e;
        phoneNumber = pN;
    }

    //method
    public void print() {
        System.out.println("Name: " + name);
        System.out.println("Email: " + email);
        System.out.println("Phone number: " + phoneNumber);
    }
}
```

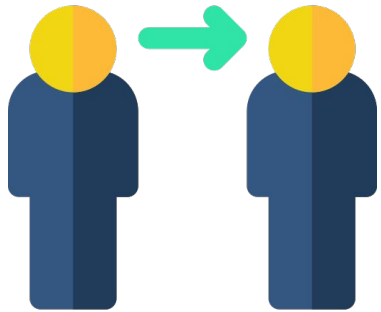
Classes and Objects

- Now we can write a main method:
  public static void main (String [] args) {
  To create an object and test out constructor and method
- First, **we need an object that is an instance of the class such as we get by calling its constructor.**
- Then **we use the dot (.) operator to call its public methods**, for example **p1.print() means call the print method on the object p1.**

```
// call the constructor to create a new person
Person p1 = new Person("Sana", "sana@gmail.com", "123-456-7890");
// call p1's print method
p1.print();
```

04

Inheritance, Abstract Classes, and Interfaces



Inheritance

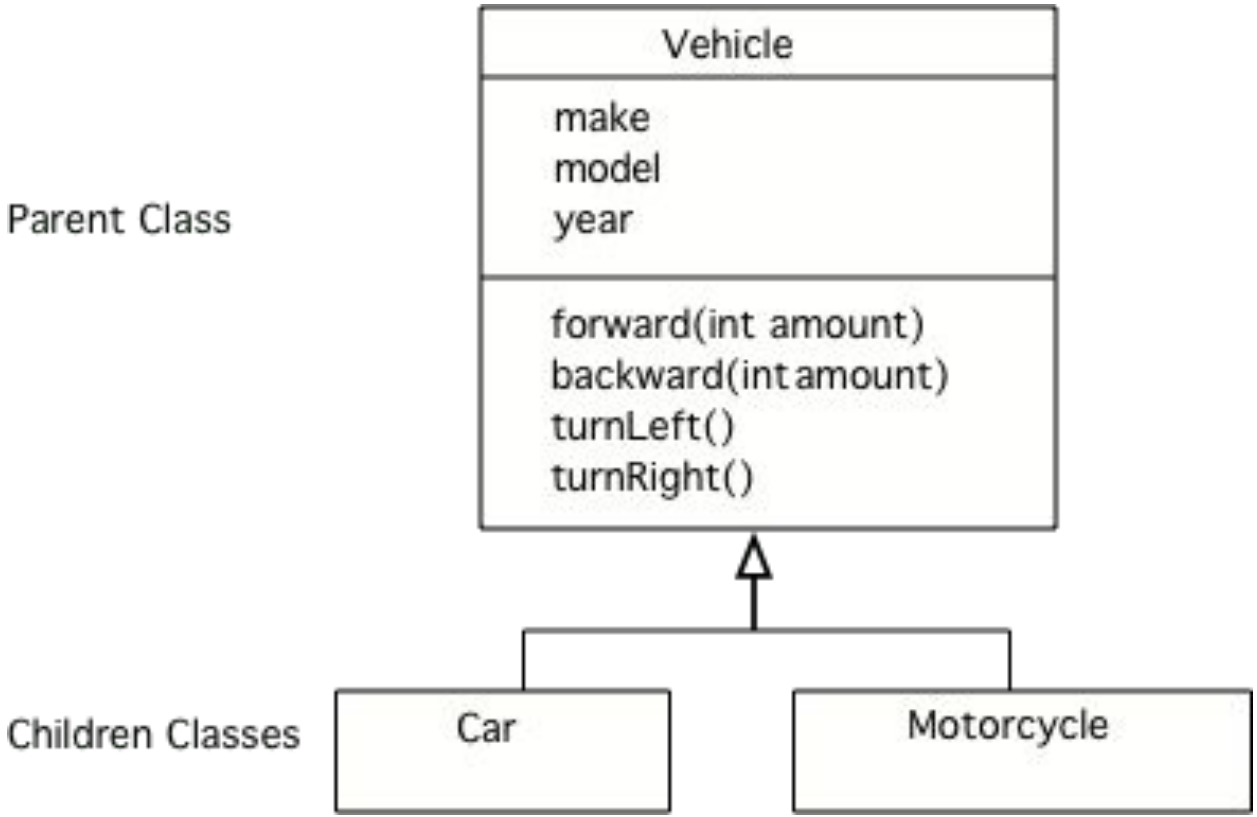
- One of the really useful features of Object-Oriented programming is **inheritance**. You may have heard of someone coming into an inheritance, which often means they were left something from a relative that died. Or, you might hear someone say that they have inherited certain traits from their parents.
- In Java, it is possible to inherit attributes and methods from one class to another.
- We group the "inheritance concept" into two categories:
  - **subclass** (child) - the class that inherits from another class
  - **superclass** (parent) - the class being inherited from

- Inheritance in Java is a mechanism in which **one object acquires all the properties and behaviors of a parent object.**

Inheritance

- The idea behind inheritance in Java is that you can **create new classes that are built upon existing classes**.
- When you inherit from an existing class, you can **reuse methods and fields of the parent class**. Moreover, **you can add new methods and fields in your current class also**.
- To inherit from a class, **use the extends keyword**.
- When one class inherits from another, we can say that it is the same kind of thing as the parent class (the class it inherits from).
- For example, a car is a kind of vehicle, a motorcycle is another kind of vehicle.
- All vehicles have a make, model, and year that they were created, can go forward, backward, turn left, and turn right.
- The following **UML (Unified Modeling Language) class diagram** shows the classes and the relationships between the classes

Inheritance



Inheritance

- As aforementioned, a parent class is specified using the extends keyword
- **Use the Java keyword extends after the class name and then followed by the parent class name** to specify the parent class as shown below:

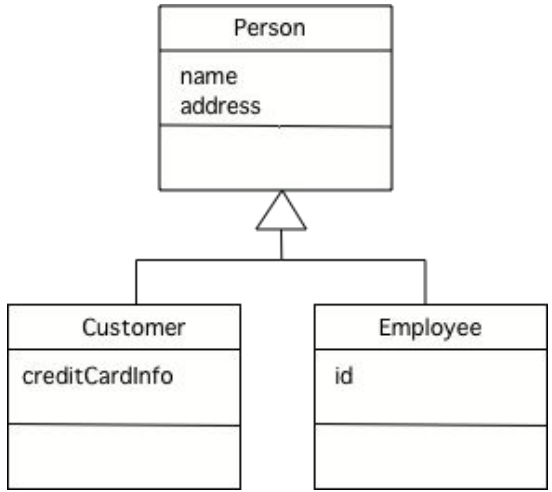
```
public class Car extends Vehicle
public class Motorcycle extends Vehicle
```

- Note: While a person has two parents, **a Java class can only inherit from one parent class.**
- If you leave off the extends keyword when you declare a class then the class will inherit from the Object class.

Inheritance

- Inheritance is useful for **generalization** in which case you may notice that several classes share the same data and/or behavior and which you can contain in a parent class.
- It is also useful for **specialization** which is when you want most of the behavior of a parent class, but want to do at least one thing differently and/or add more data.

- For example, **Customers and Employees are both people** so it makes sense use the **general Person class**.
- However, **an employee is a person but also has a unique id**.
- **A customer is a person, but also has a credit card.**
- This is an example of **specialization**.



Inheritance

- In Java, the **super** keyword is used to refer to the parent class of a subclass.
- Whenever you create the instance of subclass, **an instance of parent class is created implicitly which is referred by super reference variable**.
- super is used to call a superclass constructor, to call a superclass method, and to access a superclass field
- When calling a superclass constructor, the super() statement must be the first statement in the constructor of the subclass.

```java
public class Customer extends Person {
    //instance variables
    private String ccInfo;


    //constructor
    public Customer (String n, String e, String pN, String cc) {
        super(n, e, pN); //calls constructor of superclass Person
        ccInfo = cc;
    }

    //method
    public void printCC () {
        System.out.println("Credit card #: " + ccInfo);
    }
}
```

```java
public class Main {
    //main method
    public static void main (String[] args) {
        //create new Person obj
        Person p1 = new Person ("Sana", "sana@gmail.com", "123-456-7890");
        p1.print();
        System.out.println("-------------");
        Person p2 = new Person ("Lee", "lee@gmail.com", "123-456-7890");
        p2.print();
        System.out.println("-------------");
        Customer c1 = new Customer("Tom", "tom@aol.com", "345-678-1230", "1123 3456 2345 6789");
        c1.print(); //inherited from superclass Person
        c1.printCC(); //written in subclass Customer
    }
}
```

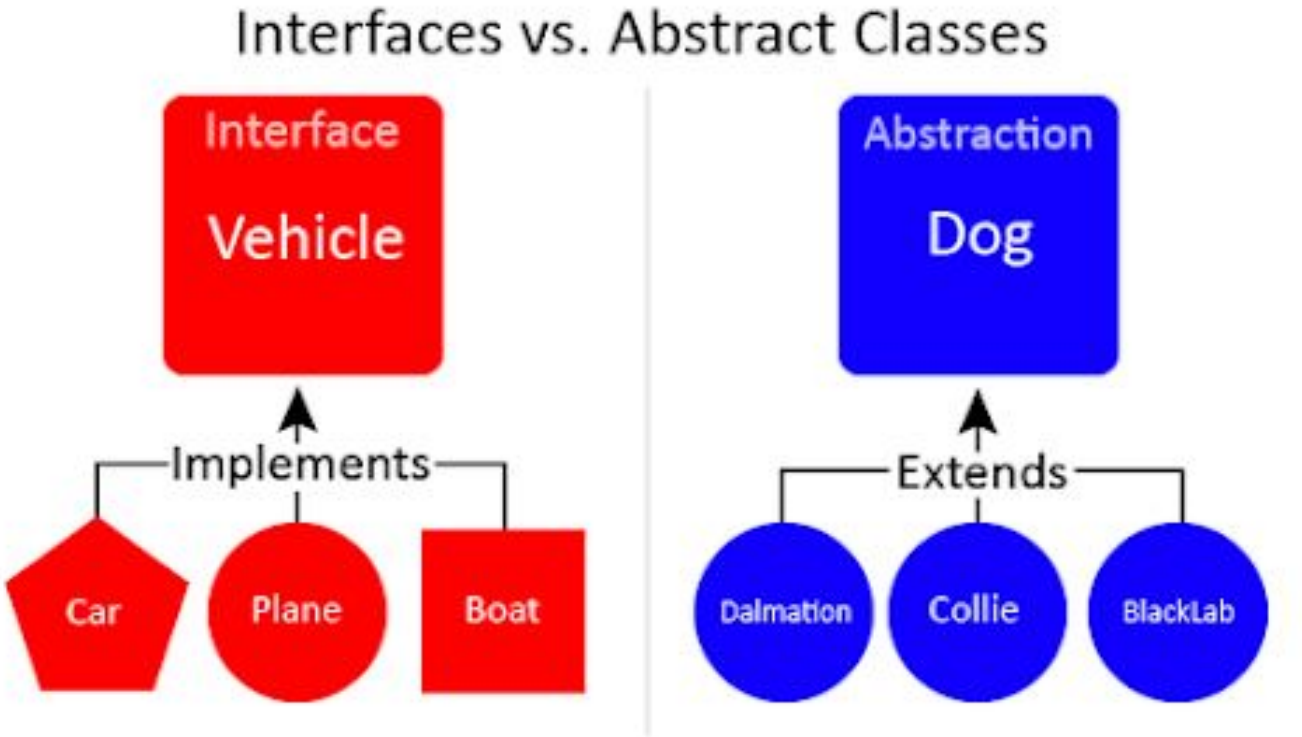
Abstraction

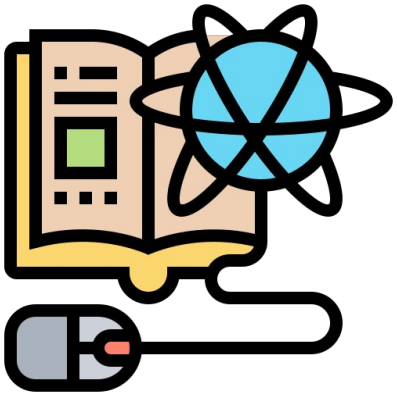
- Data **abstraction** is the process of hiding certain details and showing only essential information to the user (i.e. showing only the required features, and hiding how those features are implemented behind the scene)
- Abstraction can be achieved with either **abstract classes** or **interfaces**
- The abstract keyword is a non-access modifier, used for classes and methods:
    - **Abstract class**: is a **restricted class that cannot be used to create objects** (**to access it, it must be inherited from another class**).
    - **Abstract method**: can **only be used in an abstract class, and it does not have a body.** The body is provided by the subclass (inherited from).

Abstraction

- Another way to achieve abstraction in Java, is with interfaces.
- An **interface** is **a completely "abstract class" that is used to group related methods with empty bodies**
- To access the interface methods, t**he interface must be "implemented"** (kinda like inherited) **by another class with the implements keyword (instead of extends).**
- The body of the interface method is provided by the "implement" class
- A class **can inherit from only one abstract class**, but it **can implement multiple interfaces**.
- This is because **an abstract class represents a type of object**, while **an interface represents a set of behaviors**.

Abstraction





05

Overview and Next Steps

Overview

- At this point we have reviewed the basic Java syntax and main fundamentals
- In addition, you should have gained some familiarity with:
  - **The concept of Object-Oriented Programming vs Procedural Programming**
  - **The concept of Classes and Objects**
  - **Inheritance and Subclasses and Superclasses**
  - **The differences between an Abstract Class and an Interface**
- All of which will be covered in further depth in your upcoming 3115 course (along with more concepts as well)

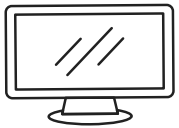
Next Steps

- You are highly encouraged to read the following text (one of the best intro books for Java AND available as a free PDF online):

Allen Downey and Chris Mayfield, *Think Java: How to Think Like a Computer Scientist*, 2nd Edition, Version 7.1.0, Green Tea Press, 2020, Creative Commons License.

- Furthermore, **tutors are available in the Learning Center – 1300 Boylan Hall** if you ever need any study assistance throughout the course

- Congratulations and best of luck on your CS journey! 🎉

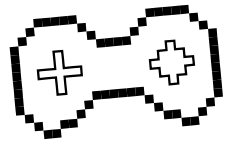


Thank You!

Presented By:

Amara Auguste

CS Tutor, Graduate Student,
and Adjunct Lecturer



Do you have any further questions?

auguste@sci.brooklyn.cuny.edu

amaraauguste.github.io