LEARN PROGRAMMING IN JAVA™

Learning OO Java™ as a Beginner – The Basics

COURSE AGENDA

- What is an Object
- The Basics
- Operators
- Control Flow
- Implementation Hiding
- Reuse
- Interfaces
- Collections
- Functional Programming
- Stream
- Exceptions
- Generics
- Arrays

TOPIC AGENDA

- Object Objects
- Primitives
- high precision numbers
- What is a Class?
- Comments
- Creating Objects
- Fields
- Methods, Arguments & Return Values
- `static` keyword
- Packages and importing other classes
- First Java Program - Lab
  - “Hello World!”

OBJECT OBJECTS

- Q - Is Java a **pure** Object-Oriented language?
- Q - Do you care? A – No
- **Primitive Types**—Java uses primitives like `int` and `long`, which are not classes. So, not every object in Java is a `Class`.
- **Statics and Static Methods**—Java allows for referencing variables and methods not tied to an object instance.
- **Wrapper Classes**—Alternatively, Java does have wrapper classes for numbers.

Java is, therefore, a primarily object-orientated language. If you want pureness, find a job using Small Talk.

PRIMITIVES

Primitive	Wrapper	Min Size	Max Size
byte	Byte	-127	127
short	Short	-32768	32768
int *32 bits	Integer	$-2^{31} - 1$	$2^{31} - 1$
long *64 bits	Long	$-2^{63} - 1$	$2^{63} - 1$
float *32 bits	Float	2^{-126}	$(2-2^{-23}) \cdot 2^{127}$
double *64 bits	Double	2^{-1074}	$(2-2^{-52}) \cdot 2^{1023}$
char	Character	Unicode 0 \u0000	65,535 \uffff
boolean	Boolean	-	-

Do I care about the size? Do I make everything an `Integer` and a `Double`?

HIGH PRECISION NUMBERS

- The primitives and wrapper classes will not suffice if you are programming in the financial field, crypto, or science fields.
- `BigIntger` and `BigDecimal` come to the rescue.
- **Arbitrary Precision:**
  - `BigInteger` can represent integers of any size, limited only by the available memory in your system. This makes it ideal for dealing with huge numbers, such as those used in cryptography, finance, or scientific computations.
- **Precision:**
  - `BigInteger` provides precise arithmetic operations, ensuring accurate results even with huge numbers.
- **Mathematical Operations:**
  - `BigInteger` allows for various mathematical operations, from addition and subtraction to multiplication, division, modular arithmetic.

COMMENTS

Single-line comments - Everything after the `//` on that line will be ignored

```
// age of the dog
int age;

// Owner's address
Address home;

// Has been seen by the vet
boolean isVaccinated;

// First and Last Name
String name;
```

Multi-line comments - Starts with a `/*` and ends with a `*/`
everything in between will be ignored

```
/**
 * Given an Address called `from`, the function returns the number of miles
 * from the provided Address to the Canine's Address
 * @param from The provided Address to calculate the oneway distance
 * @return The number of miles, as a double, between the Canine's home and the `from`
 */
public double getDistanceInMiles(Address from) {
```

- **"Never trust a comment."**
  - Abraham Lincoln

CREATING AN OBJECT —EXAMPLE

```
Canine rover = new Canine("Rover");
University nyu = new University();
Automobile ford150LuvTrucks = new Truck("LUV-TRKS");
PersonalComputer currentLaptop = new
PersonalComputer("X123 FCV");
```

What about `String` and primitives?

- Even though `String` is a class, you do not use '`new`' to create it
- Primitives are not instantiated. That is why they are primitives.

FIELDS

- **fields**—also called properties represent the attributes for that class.
  - Recall we can add primitives and classes as fields

- **Animal->Mammal->Canine**
  - What fields would a Canine have?
  - It depends actually
    - Are you modeling a pet, a patient, or a product?
      - Let's assume we are modeling a Pet.

FIELDS - EXAMPLE

```java
package com.learnprogramminginjava.basics;

public class Canine {

    String name;
    boolean isVaccinated;
    int age;
    Address home;
}
```

Fields

- fields are declared
  - `name`, `isVaccinated`, `age`, and `Address`
- The fields have **not** been assigned a value
- NOTE: While not assigned a value, the fields do have defaults.
- The programmer's best practice is **not to rely on the default.**

Bonus *– What kind of relationship does* `Canine` *have with* `Address`*: "is-a" or "has-a"?*

ASSIGNING VALUES TO FIELDS - EXAMPLE

```
Canine dog = new Canine();
dog.name = "Fido";
dog.isVaccinated = false;
dog.age = 5;
```

Fields

- fields are assigned a value when an object of that type has been declared
  - `name`, `isVaccinated`, `age`, and `Address`
- When created (aka constructed), the class or the object creating that type can assign the values.

Bonus *—The best practice is to assign functionally accurate defaults when constructing the object. Otherwise, the caller assigns the values. Do not create bogus or incorrect defaults.*

METHODS (AKA FUNCTIONS)

- methods—Are what a class does. These are the operations and functions a class exposes to clients. For instance, a `'calculateArea'` method in a `'Rectangle'` class would calculate the area of the rectangle. This is not the same as asking a class for a field value.
- argument—When invoking a method, if it requires an argument or additional information to understand how to handle the request, then the method will require an argument to be passed
- return value—Optionally, the function can perform the action and not return a value (i.e., void), or it can return a result.
  - The return value can be a primitive or class.

METHOD - EXAMPLE

What is the name of the method? The Argument List and the return value?

```
/**
 * Given an Address called `from`, the function returns the number of miles
 * from the provided Address to the Canine's Address
 * @param from The provided Address to calculate the oneway distance
 * @return The number of miles, as a double, between the Canine's home and the `from`
 */
public double getDistanceInMiles(Address from) {
    // calculate distance `from` the Canine's home address
    // Use external mapping API like Google Maps or the Phone's Mapping software
    // Implementation is not important for this example
    return 1.00;
}
```

GETTERS & SETTERS - EXAMPLE

```java
public class Canine {

    boolean isVaccinated;
    int age;
    Address home;

    String name;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
```

- fields can be assigned values through methods. The method can also return the current value of the field.
    - `name`, `isVaccinated`, `age`, and `Address`
- Typically, programmers create a setter to set the value and a getter method to retrieve the value.
- While not enforced by the compiler, it is industry best practice to use Bean notation for the getters and setters.

What is the name of the method? The Argument List and the return value?

```java
public class Canine {

    int age;
    Address home;
    boolean isVaccinated;
    String name;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public boolean isVaccinated() {
        return isVaccinated;
    }
}
```

BEAN NOTATION

Properties in JavaBeans are accessed using getter and setter methods.

- For example, the getter method for a property named name would be `getName()`.
- The setter method for the same property would be named `setName(String name)`.
- If the property is a boolean, the getter method can use the **`is`** prefix instead:

STATIC, STATIC, STATIC

CanineUtil <<utility>>
`DOG_YEAR_FACTOR = 7;`
`getDogYears(double humanAge): double` `getHumanAge(double dogAge): double`

Static methods are associated with the class and not an object.

A static field exists only once in execution, and if the value is changed, then the value is changed for all classes that have a reference to the static field.

The Utility class has only public static methods and static fields. It is not required to be instantiated. It is often used for algorithms and quick reusable code.

The best practice is for the static function **NOT** to call out and invoke external services or be detrimental to performance.

STATIC, STATIC, STATIC EXAMPLE

```
public class CanineUtils {  new *

    static public final short DOG_YEAR_FACTOR = 7;

    /**
     * Convert a human's age into Dog Years using
     * the dog year ratio of 1:7.
     *
     * @param humanAge Human's age
     * @return Human's age in terms of a dog years.
     */
    static public double getDogYears(double humanAge){
            return humanAge / DOG_YEAR_FACTOR;
    }
```

DOG_YEAR_FACTOR is a static field; all classes do not need to create the field.

*We added the final keyword so that the field cannot be modified. (*More on that later*)

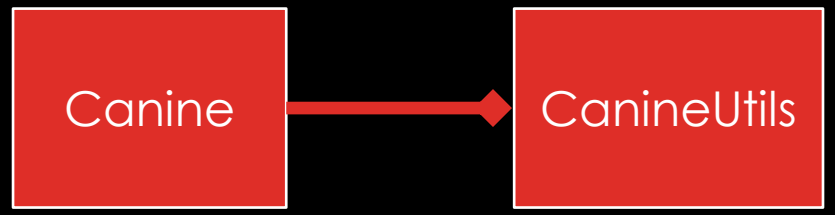
The static method belongs to the class. That is, **you do NOT use the new keyword.** Instead, prefix the method name with the class name.

```
Double dogAge = CanineUtils.getDogYears(17);
```

NOT

```
CanineUtils utils = new CanineUtils();
Double dogAge = utils.getDogYears(17);
```

IMPORTING OTHER CLASSES

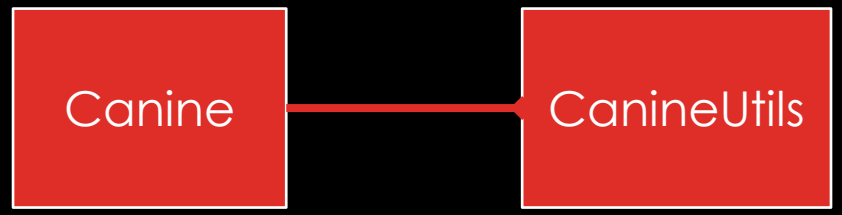


The import keyword brings (aka "import") classes or entire packages into visibility within a Java program. It allows the programmer to use them without specifying their full path.

It helps simplify code by removing the need to use fully qualified names for classes and interfaces.

Wait! What is a package?

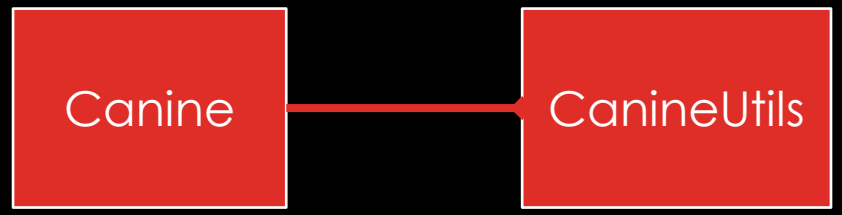
PACKAGE YOUR CLASS



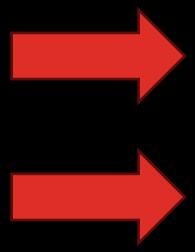
How do I package, i.e., store the classes I create in a hierarchy? How do I avoid naming conflicts? How do I visualize my design?

The package keyword defines a namespace for organizing classes, interfaces, and other **related** code. It helps to group related code logically, manage large projects more efficiently, and avoid naming conflicts by categorizing classes.

PACKAGE EXAMPLE



The package statement must be the first line in a Java source file (before any import statements or class definitions). It defines the package to which the classes in the file belong. One class can only belong to one package.



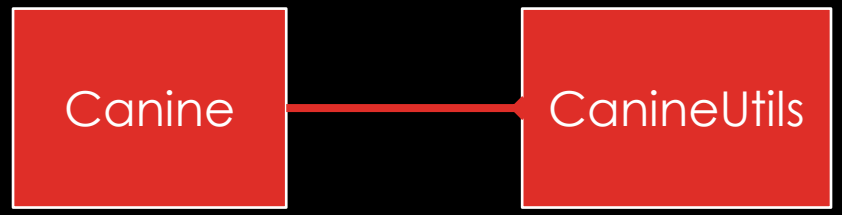
```java
package com.learnprogramminginjava.basics;

import com.learnprogramminginjava.basics.utils.CanineUtils;

public class Canine {

    int age;
    Address home;
    boolean isVaccinated;
    String name;
```

PACKAGE IN INTELLIJ

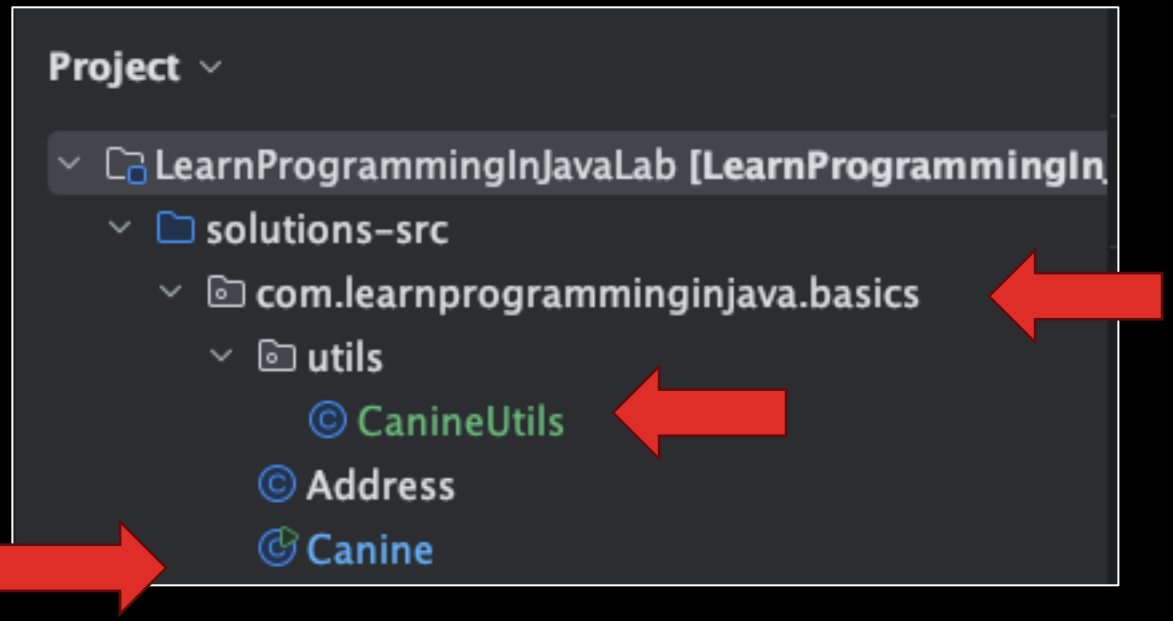


The CanineUtils class is in a package (aka directory)

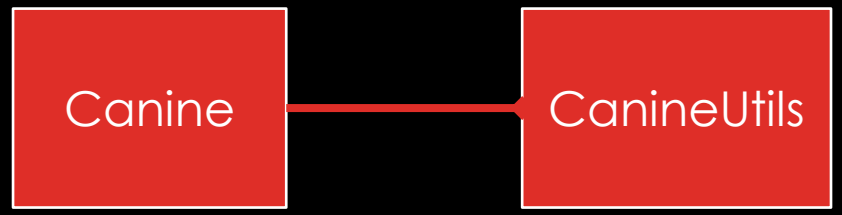
`com.learnprogramminginjava.basics.`**`utils`**

underneath the package

`com.learnprogramminginjava.basics`



PACKAGE YOUR CLASS



Note the CanineUtils class in the utils package only has the package keyword. There is no reference back to the Canine class.

```
package com.learnprogramminginjava.basics.utils;

public class CanineUtils {

    static public final short DOG_YEAR_FACTOR = 7;
```

TOPIC SUMMARY

- Object Objects
- Primitives
- high precision numbers
- What is a Class?
- Comments
- Creating Objects
- Fields
- Methods, Arguments & Return Values
- `static` keyword
- Packages and importing other classes