Lecture 2

Defining and Using Classes

CS61B, Spring 2024 @ UC Berkeley

Slides credit: Josh Hug

Defining and Instantiating Classes

Lecture 2, CS61B, Spring 2024

Coding Demo: Dog class

Dog.java

```java
public class Dog {
  public static void makeNoise() {
     System.out.println("Bark!");
  }





}
```

```
Error: Main method not found in class Dog
```

Coding Demo: Dog class

Dog.java

```java
public class Dog {
  public static void makeNoise() {
     System.out.println("Bark!");
  }

  public static void main(String[] args) {
     makeNoise();
  }
}
```

```
Bark!
```

Coding Demo: Dog class

Dog.java

```
public class Dog {
  public static void makeNoise() {
     System.out.println("Bark!");
  }

  public static void main(String[] args) {
     makeNoise();
  }
}
```

DogLauncher.java

```
public class DogLauncher {



}
```

```
Bark!
```

Coding Demo: Dog class

Dog.java

```java
public class Dog {
  public static void makeNoise() {
     System.out.println("Bark!");
  }





}
```

DogLauncher.java

```java
public class DogLauncher {
  public static void main(String[] args) {
     Dog.makeNoise();
  }
}
```

```
Bark!
```

Dog

As we saw last time:

- Every method (a.k.a. function) is associated with some class.
- To run a class, we must define a main method.
    - Not all classes have a main method!

Unlike python, there's no need to import if the two files are in the same project.

```java
public class Dog {
   public static void makeNoise() {
      System.out.println("Bark!");
   }
}
```

Can't be run directly, since there is no main method.

```java
public class DogLauncher {
   public static void main(String[] args) {
      Dog.makeNoise();
   }
}
```

Calls a method from another class. Can think of this as a class that tests out the Dog class.

Object Instantiation

Not all dogs are equal!

A Not So Good Approach

We could create a separate class for every single dog out there, but this is going to get redundant in a hurry.

```
public class MayaTheDog {
   public static void makeNoise() {
      System.out.println("aroooooooooo!");
   }
}
```

```
public class YapsterTheDog {
   public static void makeNoise() {
      System.out.println("awawawwwawwa awawaw");
   }
}
```

Object Instantiation

Classes can contain not just functions (a.k.a. methods), but also data.

- For example, we might add a `size` variable to each `Dog`.

These instances are also called 'objects'

Classes can be instantiated as objects.

- We'll create a single `Dog` class, and then create instances of this `Dog`.
- The class provides a blueprint that all `Dog` objects will follow.
    - For the example above, all `Dog` objects will have a `size`.

Let's try this out.

Coding Demo: Dog class

Dog.java

```
public class Dog {






    public static void makeNoise() {
        System.out.println("Bark!");







    }
}
```

DogLauncher.java

```
public class DogLauncher {
  public static void main(String[] args) {
      Dog.makeNoise();














  }
}
```

Coding Demo: Dog class

Dog.java

```
public class Dog {
    int weightInPounds;






    public static void makeNoise() {
        System.out.println("Bark!");







    }
}
```

DogLauncher.java

```
public class DogLauncher {
   public static void main(String[] args) {
      Dog.makeNoise();














   }
}
```

Coding Demo: Dog class

Dog.java

```java
public class Dog {
   int weightInPounds;






   public static void makeNoise() {
      if (weightInPounds < 10) {
         System.out.println("yip!");
      } else if (weightInPounds < 30) {
         System.out.println("bark.");
      } else {
         System.out.println("woooooof!");
      }
   }
}
```

```
Error: Non-static variable weightInPounds
cannot be referenced from a static context.
```

DogLauncher.java

```java
public class DogLauncher {
  public static void main(String[] args) {
     Dog.makeNoise();













  }
}
```

Coding Demo: Dog class

Dog.java

```
public class Dog {
   int weightInPounds;





   public void makeNoise() {
      if (weightInPounds < 10) {
         System.out.println("yip!");
      } else if (weightInPounds < 30) {
         System.out.println("bark.");
      } else {
         System.out.println("wooooof!");
      }
   }
}
```

DogLauncher.java

```
public class DogLauncher {
  public static void main(String[] args) {
     Dog.makeNoise();











  }
}
```

```
Error: Non-static method makeNoise cannot
be referenced from a static context.
```

Coding Demo: Dog class

Dog.java

```java
public class Dog {
   int weightInPounds;





   public void makeNoise() {
      if (weightInPounds < 10) {
         System.out.println("yip!");
      } else if (weightInPounds < 30) {
         System.out.println("bark.");
      } else {
         System.out.println("wooooof!");
      }
   }
}
```

DogLauncher.java

```java
public class DogLauncher {
  public static void main(String[] args) {
     Dog d = new Dog();
     d.weightInPounds = 25;
     d.makeNoise();









  }
}
```

```
bark.
```

Coding Demo: Dog class

Dog.java

```java
public class Dog {
   int weightInPounds;





   public void makeNoise() {
      if (weightInPounds < 10) {
         System.out.println("yip!");
      } else if (weightInPounds < 30) {
         System.out.println("bark.");
      } else {
         System.out.println("wooooooof!");
      }
   }
}
```

DogLauncher.java

```java
public class DogLauncher {
  public static void main(String[] args) {
     Dog d = new Dog();
     d.weightInPounds = 51;
     d.makeNoise();
  }
}
```

```
wooooooof!
```

Coding Demo: Dog class

Dog.java

```java
public class Dog {
   int weightInPounds;

   public Dog(int w) {
      weightInPounds = w;
   }

   public void makeNoise() {
      if (weightInPounds < 10) {
         System.out.println("yip!");
      } else if (weightInPounds < 30) {
         System.out.println("bark.");
      } else {
         System.out.println("woooooof!");
      }
   }
}
```

DogLauncher.java

```java
public class DogLauncher {
  public static void main(String[] args) {
     Dog d = new Dog(51);
     d.makeNoise();




  }
}
```

```
woooooof!
```

Our Dog Class

```
public class Dog {
   public int weightInPounds;

   public Dog(int startingWeight) {
      weightInPounds = startingWeight;
   }

   public void makeNoise() {
      if (weightInPounds < 10) {
         System.out.println("yipyipyip!");
      } else if (weightInPounds < 30) {
         System.out.println("bark. bark.");
      } else {
         System.out.println("woof!");
      }
   }
}
```

Java vs. Python Classes

```java
public class Dog {
  public int weightInPounds;

  public Dog(int startingWeight) {
     weightInPounds = startingWeight;
  }

  public void makeNoise() {
     if (weightInPounds < 10) {
        System.out.println("yipyipyip!");
     } else if (weightInPounds < 30) {
        System.out.println("bark. bark.");
     } else {
        System.out.println("woof!");
     }
  }
}
```

For those of you who know Python, the equivalent code is given below.

```python
class Dog():
  def __init__(self, startingWeight):
    self.weightInPounds = startingWeight

  def makeNoise(self):
    if self.weightInPounds < 10:
      print "yipyipyip!"
    elif self.weightInPounds < 30:
      print "bark. Bark."
    else:
      print "woof!"
```

Class Terminology

Lecture 2, CS61B, Spring 2024

Defining a Typical Class (Terminology)

```
public class Dog {
    public int weightInPounds;

    public Dog(int startingWeight) {
        weightInPounds = startingWeight;
    }

    public void makeNoise() {
        if (weightInPounds < 10) {
            System.out.println("yipyipyip!");
        } else if (weightInPounds < 30) {
            System.out.println("bark. bark.");
        } else {
            System.out.println("woof!");
        }
    }
}
```

Instance variable. Can have as many of these as you want.

Constructor (similar to a method, but not a method). Determines how to instantiate the class.

Non-static method, a.k.a. Instance Method. Idea: If the method is going to be invoked by an instance of the class (as in the next slide), then it should be non-static.

Roughly speaking: If the method needs to use "**my**" instance variables", the method must be non-static.

Object Instantiation

Classes can contain not just functions (a.k.a. methods), but also data.

- For example, we might add a `size` variable to each `Dog`.

These instances are also called ‘objects’

Classes can be instantiated as objects.

- We’ll create a single `Dog` class, and then create instances of this `Dog`.
- The class provides a blueprint that all `Dog` objects will follow.
    - For the example above, all `Dog` objects will have a `size`.
    - **Cannot add new instance variables to a Dog. They must ALL obey the blueprint exactly.**

```java
public class DogLauncher {
   public static void main(String[] args) {
      Dog hugeDog = new Dog(150);
      hugeDog.weightInPounds = 5; // guaranteed to exist
      hugeDog.name = "frank"; // syntax error!
   }
}
```

Instantiating a Class and Terminology

```
public class DogLauncher {
   public static void main(String[] args) {
      Dog smallDog;
      new Dog(20);
      smallDog = new Dog(5);
      Dog hugeDog = new Dog(150);
      smallDog.makeNoise();
      hugeDog.makeNoise();
   }
}
```

Declaration of a Dog variable.

Instantiation of the Dog class as a Dog Object.

Instantiation and **Assignment**.

Declaration, Instantiation and **Assignment**.

Invocation of the 150 lb Dog's makeNoise method.

The dot notation means that we want to use a method or variable belonging to hugeDog, or more succinctly, a ***member*** of hugeDog.

Arrays of Objects

To create an array of objects:

- First use the **new** keyword to create the array.
- Then use **new** again for each object that you want to put in the array.

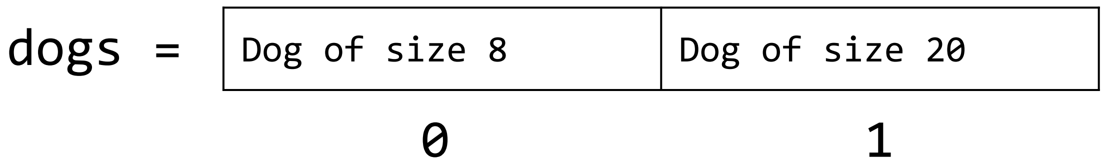
Example:

```
Dog[] dogs = new Dog[2];
dogs[0] = new Dog(8);
dogs[1] = new Dog(20);
dogs[0].makeNoise();
```

Creates an array of Dogs of size 2.

Yipping occurs.

After code runs:



Static vs. Instance Members

Lecture 2, CS61B, Spring 2024

Static vs. Non-Static

Key differences between static and non-static (a.k.a. instance) methods:

- Static methods are invoked using the class name, e.g. Dog.makeNoise();
- Instance methods are invoked using an instance name, e.g. maya.makeNoise();
- Static methods can't access "my" instance variables, because there is no "me".

Non-static

```
public void makeNoise() {
   if (weightInPounds < 10) {
      System.out.println("yipyipyip!");
   } else if (weightInPounds < 30) {
      System.out.println("bark. bark.");
   } else { System.out.println("woof!"); }
}
```

Static

```
public static void makeNoise() {
   System.out.println("Bark!");
}
```

This method cannot access weightInPounds!

Invocation:

```
Dog.makeNoise();
```

Invocation:

```
Dog maya = new Dog(100);
maya.makeNoise();
```

Why Static Methods?

Some classes are never instantiated. For example, Math.

- `x = Math.round(5.6);`

Much nicer than:

```
Math m = new Math();
x = m.round(x);
```

Sometimes, classes may have a mix of static and non-static methods, e.g.

```
public static Dog maxDog(Dog d1, Dog d2) {
   if (d1.weightInPounds > d2.weightInPounds) {
      return d1;
   }
   return d2;
}
```

Coding Demo: maxDog

Dog.java

```
public class Dog {
  int weight;
  public Dog(int w) { ... }
  public void makeNoise() { ... }




















}
```

DogLauncher.java

```
public class DogLauncher {
  public static void main(String[] args) {
     Dog d = new Dog(15);
     d.makeNoise();


















  }
}
```

Coding Demo: maxDog

Dog.java

```
public class Dog {
  int weight;
  public Dog(int w) { ... }
  public void makeNoise() { ... }
















}
```

DogLauncher.java

```
public class DogLauncher {
  public static void main(String[] args) {
     Dog d = new Dog(15);
     Dog d2 = new Dog(100);

     Dog bigger = Dog.maxDog(d, d2);
     bigger.makeNoise();













  }
}
```

Coding Demo: maxDog

Dog.java

```java
public class Dog {
  int weight;
  public Dog(int w) { ... }
  public void makeNoise() { ... }


  public static Dog maxDog(Dog d1, Dog d2) {
     if (d1.weight > d2.weight) {
        return d1;
     }
     return d2;
  }







}
```

DogLauncher.java

```java
public class DogLauncher {
  public static void main(String[] args) {
     Dog d = new Dog(15);
     Dog d2 = new Dog(100);

     Dog bigger = Dog.maxDog(d, d2);
     bigger.makeNoise();











  }
}
```

Coding Demo: maxDog

Dog.java

```java
public class Dog {
  int weight;
  public Dog(int w) { ... }
  public void makeNoise() { ... }


  public static Dog maxDog(Dog d1, Dog d2) {
     if (d1.weight > d2.weight) {
        return d1;
     }
     return d2;
  }

  public Dog maxDog(Dog d2) {
     if (weight > d2.weight) {
        return this;
     }
     return d2;
  }
}
```

DogLauncher.java

```java
public class DogLauncher {
  public static void main(String[] args) {
     Dog d = new Dog(15);
     Dog d2 = new Dog(100);

     Dog bigger = Dog.maxDog(d, d2);
     bigger.makeNoise();











  }
}
```

Coding Demo: maxDog

Dog.java

```java
public class Dog {
  int weight;
  public Dog(int w) { ... }
  public void makeNoise() { ... }


  public static Dog maxDog(Dog d1, Dog d2) {
     if (d1.weight > d2.weight) {
        return d1;
     }
     return d2;
  }

  public Dog maxDog(Dog d2) {
     if (weight > d2.weight) {
        return this;
     }
     return d2;
  }
}
```

DogLauncher.java

```java
public class DogLauncher {
  public static void main(String[] args) {
     Dog d = new Dog(15);
     Dog d2 = new Dog(100);

     Dog bigger = Dog.maxDog(d, d2);
     bigger.makeNoise();

     bigger = d.maxDog(d2);
     bigger.makeNoise();








  }
}
```

Coding Demo: maxDog

Dog.java

```
public class Dog {
  int weight;
  public Dog(int w) { ... }
  public void makeNoise() { ... }
  public static String binomen = "canis";

  public static Dog maxDog(Dog d1, Dog d2) {
     if (d1.weight > d2.weight) {
        return d1;
     }
     return d2;
  }

  public Dog maxDog(Dog d2) {
     if (weight > d2.weight) {
        return this;
     }
     return d2;
  }
}
```

DogLauncher.java

```
public class DogLauncher {
  public static void main(String[] args) {
     Dog d = new Dog(15);
     Dog d2 = new Dog(100);

     Dog bigger = Dog.maxDog(d, d2);
     bigger.makeNoise();

     bigger = d.maxDog(d2);
     bigger.makeNoise();

     System.out.println(Dog.binomen);
  }
}
```

Static Variables (are Dangerous)

Classes can also have static variables.

- You should always access class variables using the class name, not an instance name.
  - Bad coding style to do something like maya.binomen.
  - Even worse to do something like maya.binomen = “Vulpes vulpes”
- **Warning: Strongly recommended to avoid static variables whose values change.**
  - Leads to complicated code: Becomes hard to mentally keep track of which parts of your program read and write from/to the static variable. For more read this.

```
public class Dog {
   public int weightInPounds;
   public static String binomen = "Canis familiaris";

   public Dog(int startingWeight) {
      weightInPounds = startingWeight;
   }
   ...
}
```

Never changes. It’s a constant.

Static vs. Non-Static

A class may have a mix of static and non-static members.

- A variable or method defined in a class is also called a member of that class.
- Static members are accessed using class name, e.g. Dog.binomen.
- Non-static members cannot be invoked using class name: ~~Dog.makeNoise()~~
- Static methods must access instance variables via a specific instance, e.g. d1.

```java
public class Dog {
   public int weightInPounds;
   public static String binomen = "Canis familiaris";

   public Dog(int startingWeight) {
      weightInPounds = startingWeight;
   }

   public static Dog maxDog(Dog d1, Dog d2) {
      if (d1.weightInPounds > d2.weightInPounds)
         { return d1; }
      return d2;
   }
   ...
```

```java
   ...
   public void makeNoise() {
      if (weightInPounds < 10) {
         System.out.println("yipyipyip!");
      } else if (weightInPounds < 30) {
         System.out.println("bark. bark.");
      } else {
         System.out.println("woof!");
      }
   }
}
```

Practice Question

Lecture 2, CS61B, Spring 2024

Question: Will this program compile? If so, what will it print?

```
public class DogLoop {
   public static void main(String[] args) {
       Dog smallDog = new Dog(5);
       Dog mediumDog = new Dog(25);
       Dog hugeDog = new Dog(150);

       Dog[] manyDogs = new Dog[4];
       manyDogs[0] = smallDog;
       manyDogs[1] = hugeDog;
       manyDogs[2] = new Dog(130);

       int i = 0;
       while (i < manyDogs.length) {
           Dog.maxDog(manyDogs[i], mediumDog).makeNoise();
           i = i + 1;
       }
   }
}
```

< 10: yip

< 30: bark

>=30: woof

Answer to Question

Won’t go over in live lecture.

Use the Java visualizer to see the solution here: http://goo.gl/HLzN6s

Video solution: https://www.youtube.com/watch?v=Osuy8UEH03M

Interactive Debugging

Lecture 2, CS61B, Spring 2024

Interactive Debugging

So far (e.g. in CS61A), you might have added print statements to find bugs in your code.

Today, we'll use IntelliJ's built-in, interactive debugging tool to find bugs in some code.

Goal: Larger Than Four Neighbors

Lecture 2, CS61B, Spring 2024

Goal: largerThanFourNeighbors

Suppose we want to write a method:

```
public static Dog[] largerThanFourNeighbors(Dog[] dogs)
```

This method will return a new array that contains every Dog that is larger than its 4 closest neighbors, i.e. the two on the left and the two in the right.

If there are not enough neighbors, i.e. you're at the end of the array, then consider just the neighbors that exist.

For example:

- Input: Dogs with size `[10, 20, `**`30`**`, 25, 20, `**`40`**`, 10]`
- Returns: Dogs with size `[30, 40]`.
  - 30 is greater than 10, 20, 25, and 20.
  - 40 is greater than 25, 20, and 10.

Goal: largerThanFourNeighbors

Suppose we want to write a method:

```
public static Dog[] largerThanFourNeighbors(Dog[] dogs)
```

If input Dog sizes are `[10, 15, 20, 15, 10, 5, 10, 15, 22, 20]`, what will be the size of the Dogs returned?

```
A.  [20]
B.  [20, 22]
C.  [20, 22, 20]
```

Goal: largerThanFourNeighbors

Suppose we want to write a method:

```
public static Dog[] largerThanFourNeighbors(Dog[] dogs)
```

If input Dog sizes are [`10, 15,` **`20,`** `15, 10, 5, 10, 15,` **`22,`** `20`], what will be the size of the Dogs returned?

A. `[20]`

B. `[20, 22]`

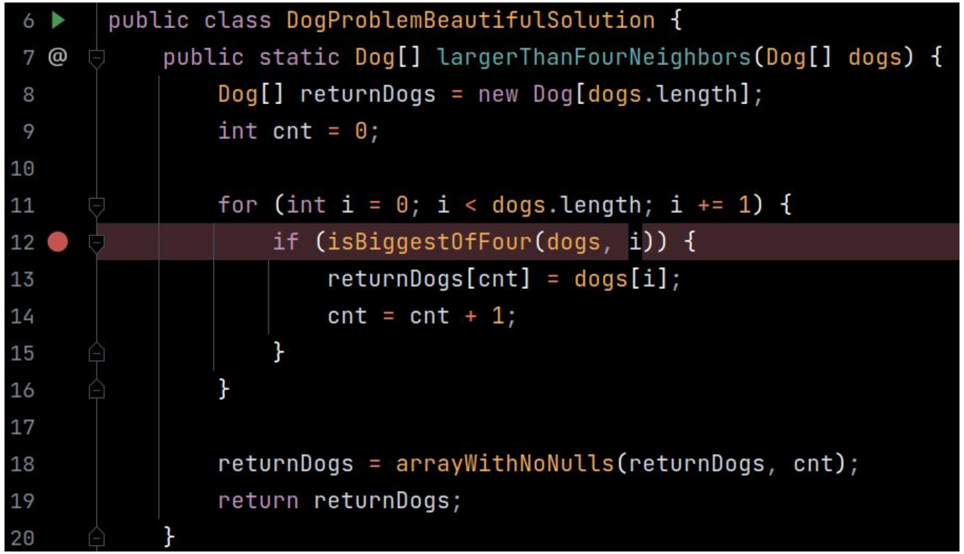
C. `[20, 22, 20]`

Using the Debugger

Lecture 2, CS61B, Spring 2024

Setting Breakpoints

- Breakpoints: Places in the code where the debugger will pause and let you inspect the program state
- In IntelliJ:
    - To set/unset breakpoints, click just to the right of the line number
    - Breakpoints are highlighted in red
    - Click to launch the debugger and run the program, pausing at breakpoints

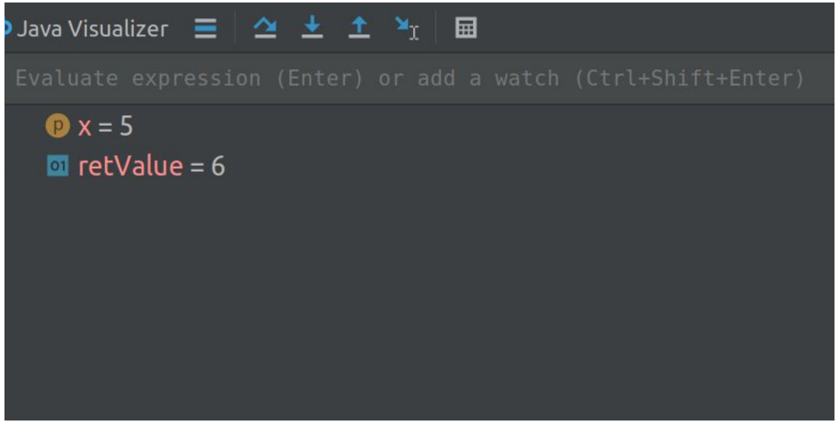


```java
public class DogProblemBeautifulSolution {
    public static Dog[] largerThanFourNeighbors(Dog[] dogs) {
        Dog[] returnDogs = new Dog[dogs.length];
        int cnt = 0;

        for (int i = 0; i < dogs.length; i += 1) {
            if (isBiggestOfFour(dogs, i)) {
                returnDogs[cnt] = dogs[i];
                cnt = cnt + 1;
            }
        }

        returnDogs = arrayWithNoNulls(returnDogs, cnt);
        return returnDogs;
    }
```

Inspecting Program State

- When the program is paused, you can view the values of all the variables (as if you had added print statements)
- You can also execute lines of code interactively in the "Evaluate expression" box

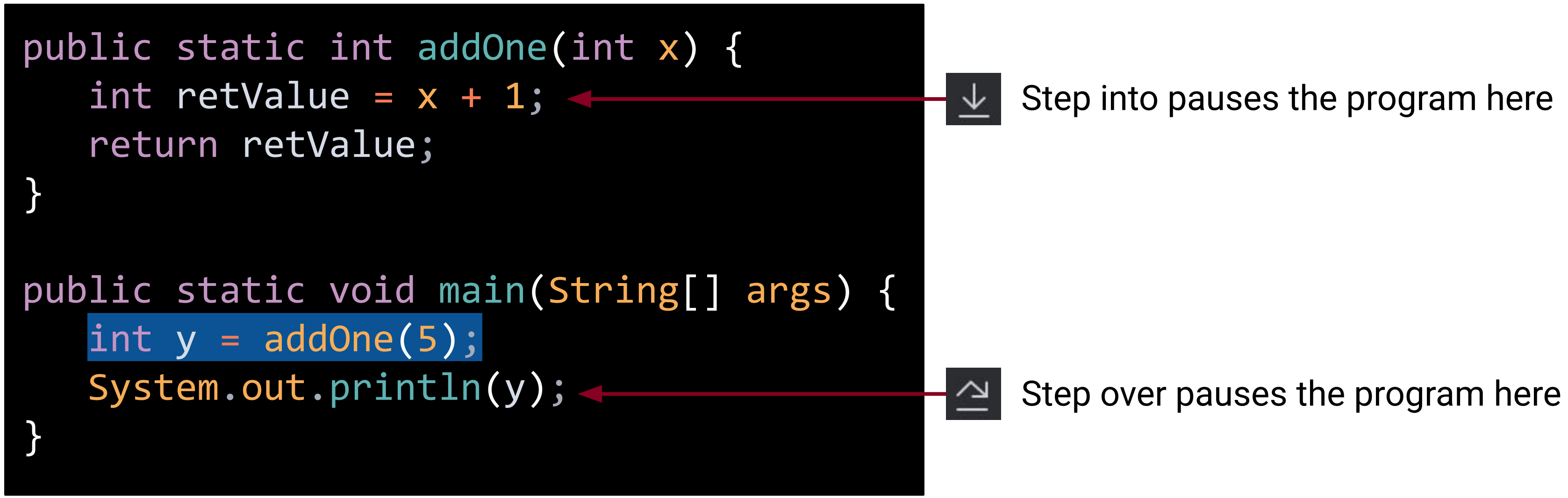
```java
public static int addOne(int x) {
   int retValue = x + 1;
   return retValue;
}

public static void main(String[] args) {
   int y = addOne(5);
   System.out.println(y);
}
```



```
Evaluate expression (Enter) or add a watch (Ctrl+Shift+Enter)
x = 5
retValue = 6
```

Stepping Over vs. Stepping In

- IntelliJ highlights the line about to execute (has not executed yet)
- If the highlighted line contains a function call:
    - steps *over* the function call, and pauses at the next line after calling the function
      Useful if you don't care about the code inside the function
    - steps *into* the function call, and pauses at the first line of the function
      Useful if you want to step through the code inside the function



```java
public static int addOne(int x) {
    int retValue = x + 1;
    return retValue;
}

public static void main(String[] args) {
    int y = addOne(5);
    System.out.println(y);
}
```

Step into pauses the program here
Step over pauses the program here

Continue

- You can set multiple breakpoints
- resumes running the program, pausing at the next breakpoint encountered

```java
public static Dog[] largerThanFourNeighbors(Dog[] dogs) {
    Dog[] returnDogs = new Dog[dogs.length];
    int cnt = 0;

    for (int i = 0; i < dogs.length; i += 1) {
        if (isBiggestOfFour(dogs, i)) {
            returnDogs[cnt] = dogs[i];
            cnt = cnt + 1;
        }
    }

    returnDogs = arrayWithNoNulls(returnDogs, cnt);
    return returnDogs;
}
```

If we're currently paused here...

will run the entire for loop, and pause the program at this breakpoint