Announcements

I’ve created a checklist of 61B tasks for weeks 1 to 5:

- https://docs.google.com/document/d/1lzEGT8g4SmNQsfoXeSxj6kja0MhYr6x1-IH5WDtxPMQ/edit?tab=t.0

If you joined the class late, please see:

- https://sp25.datastructur.es/policies/#late-adding-cs61b

HW0A is due today.

- Sorry about the HW0A chaos, the PracticeIt site that we started using in Spring 2023 was shut down November 1st.
  - We should have realized it was down!
  - If you need some extra time to get set up to work on HW0A, please feel free to use beacon to request an extension.

Further Notes for Webcast Viewers

- Any time I’m live coding, I advise you to pause frequently and try to anticipate my next move. You’ll probably learn more by trying to guess what I’m going to do rather than just watching me do it.

Lecture 2

Defining Classes. Lists and Maps.

CS61B, Spring 2025 @ UC Berkeley

Josh Hug and Justin Yokota

Defining and Instantiating Classes

Lecture 2, CS61B, Spring 2025

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

```
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

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

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

DogLauncher.java

```java
public class DogLauncher {



}
```

```
Bark!
```

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();
  }
}
```

```
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.

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

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

```
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("arooooooooooo!");
   }
}
```

```
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. See the webcast video or the following hidden slides for the step-by-step process of constructing the Dog class.

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

```java
public class Dog {
   int weightInPounds;





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







   }
}
```

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

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

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 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.makeNoise();













  }
}
```

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

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 d = new Dog();
     d.weightInPounds = 25;
     d.makeNoise();










  }
}
```

```
bark.
```

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("woooooof!");
      }
   }
}
```

DogLauncher.java

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











  }
}
```

```
woooooof!
```

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

```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!");
      }
   }
}
```

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 2023

Defining a Typical Class (Terminology)

```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!");
        }
    }
}
```

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

```java
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.

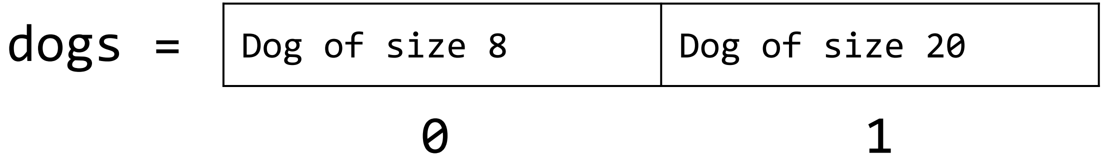
Example:

```java
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 2023

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

```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!"); }
}
```

Static

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

This method cannot access weightInPounds!

Invocation:

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

Invocation:

```java
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

```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

```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.

```
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;
   }
   ...
```

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

Puzzle: 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

Lists in Java 4.0

Lecture 2, CS61B, Spring 2023

Lists

In programming languages, a list is an ordered sequence of objects, often represented by comma-separated values in-between brackets.

Example: [3, 6, 9, 12, 15]

Lists support a variety of operations which vary according to the whims of the authors who wrote the code for the list. Some examples:

- Append an item, e.g. we could append 18 to the list above.
- Retrieve an item by index, e.g. we could ask for the 0th item and get 3.
- Removing an item by index or value, e.g. we could remove the 9 from the list.

For more, see wikipedia: https://en.wikipedia.org/wiki/List_(abstract_data_type)

Lists in Python

Python lists have very simple syntax, given below.

```
L = []
L.append("a")
L.append("b")
L.append("c")
print(L)
```

```
['a', 'b', 'c']
```

Lists in Java

Let’s try to make a copy of the code below using IntelliJ.

- See the recording for more discussion. These slides will be minimalist.

```
L = []
L.append("a")
L.append("b")
L.append("c")
print(L)
```

```
['a', 'b', 'c']
```

Lists in Java Attempt #1

Let’s try to make a copy of the code below using IntelliJ.

- The Java code below won’t compile.
- IntelliJ gives the error “can’t resolve symbol List”.
- The fix is to import List.

```
L = []
L.append("a")
L.append("b")
L.append("c")
print(L)
```

```
['a', 'b', 'c']
```

```
public class ListDemo {
    public static void main(String[] args) {
        List L = new List();
    }
}
```

Can either add import statement to code, or use the IntelliJ option-enter or alt-enter hotkey.

Lists in Java Attempt #2

Let’s try to make a copy of the code below using IntelliJ.

- The code below also won’t compile.
- This time IntelliJ complains that “List is abstract, cannot be instantiated”.
- The problem is that we need to pick a specific type of List to create.
  - (More in a moment!)

```
L = []
L.append("a")
L.append("b")
L.append("c")
print(L)
```

```
['a', 'b', 'c']
```

```
import java.util.List;

public class ListDemo {
    public static void main(String[] args) {
        List L = new List();
    }
}
```

Lists in Java Attempt #3

This code finally compiles.

- Note: This code is very old school Java (circa 2004), and we'll update it to be more modern a bit later.
- Now that we have a list, we can start adding things to it.

```
L = []
L.append("a")
L.append("b")
L.append("c")
print(L)
```

```
['a', 'b', 'c']
```

```
import java.util.List;
import java.util.ArrayList;

public class ListDemo {
    public static void main(String[] args) {
        List L = new ArrayList();
    }
}
```

We'll talk about this distinction shortly.

Lists in Java Attempt #3

We've written the equivalent Java program!

Now let's reflect on that distinction between List and ArrayList.

```
L = []
L.append("a")
L.append("b")
L.append("c")
print(L)
```

```
['a', 'b', 'c']
```

```
import java.util.List;
import java.util.ArrayList;

public class ListDemo {
    public static void main(String[] args) {
        List L = new ArrayList();
        L.add("a");
        L.add("b");
        L.add("c");
        System.out.println(L);
```

Abstract Data Types vs. Concrete Implementations

Lecture 2, CS61B, Spring 2023

Alternate Types of List

Java has other types of Lists. Let's take a peek:

- https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/List.html
- Most common type of **List** is **ArrayList**, but **LinkedList** is also used sometimes.

```
L = []
L.append("a")
L.append("b")
L.append("c")
print(L)
```

```
['a', 'b', 'c']
```

```
import java.util.List;
import java.util.ArrayList;

public class ListDemo {
    public static void main(String[] args) {
        List x = new ArrayList();
        ...
    }
}
```

Alternate Types of List

Java has other types of Lists. Let's take a peek:

- https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/List.html
- Most common type of **List** is **ArrayList**, but **LinkedList** is also used sometimes.

```
L = []
L.append("a")
L.append("b")
L.append("c")
print(L)
```

```
['a', 'b', 'c']
```

```
import java.util.List;
import java.util.ArrayList;

public class ListDemo {
    public static void main(String[] args) {
        List x = new LinkedList();
        ...
    }
}
```

List

In Python, there is no distinction between the abstract idea of a list and an actual list.

- A list is a list is a list. Programmer has no choice of what type of list.

In Java, there are many types of lists.

- Programmer has to decide which one.

```
L = []
L.append("a")
L.append("b")
L.append("c")
print(L)
```

```
['a', 'b', 'c']
```

```
import java.util.List;
import java.util.ArrayList;

public class ListDemo {
    public static void main(String[] args) {
        List x = new LinkedList();
        ...
    }
```

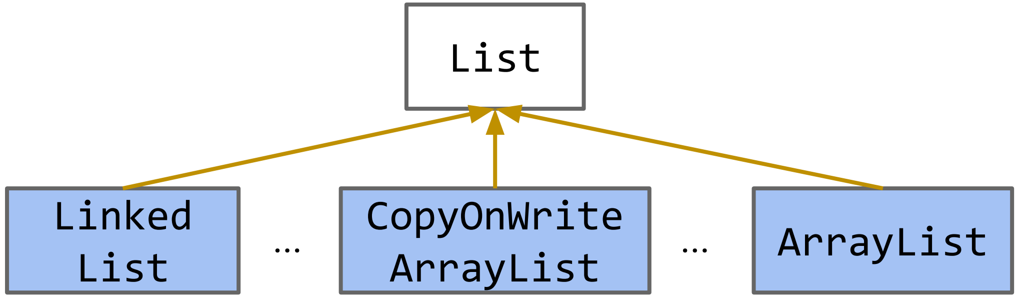
Abstract Data Types vs. Concrete Implementations

Another term used for List in Java is “Abstract Data Type”.

- Any list is guaranteed to have at least the operations in https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/List.html.

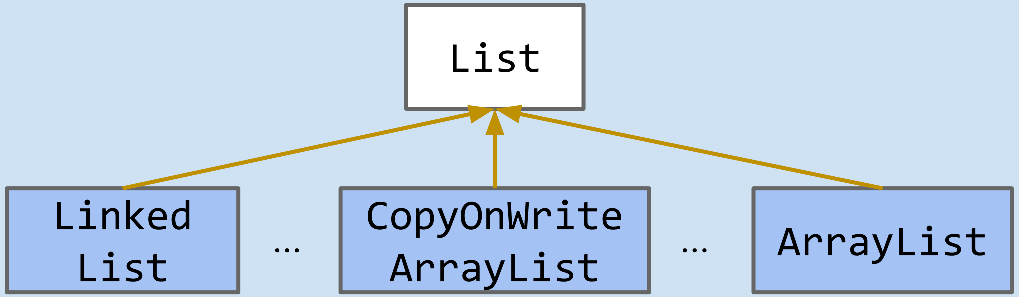
Each implementation, e.g. LinkedList, is known as a “Concrete Implementation”.

- Code for different types of list may be radically different.
- All concrete implementations have at least the operations guaranteed by every List (at the link above).



Abstract Data Types

Why bother having multiple implementations? What do you think?

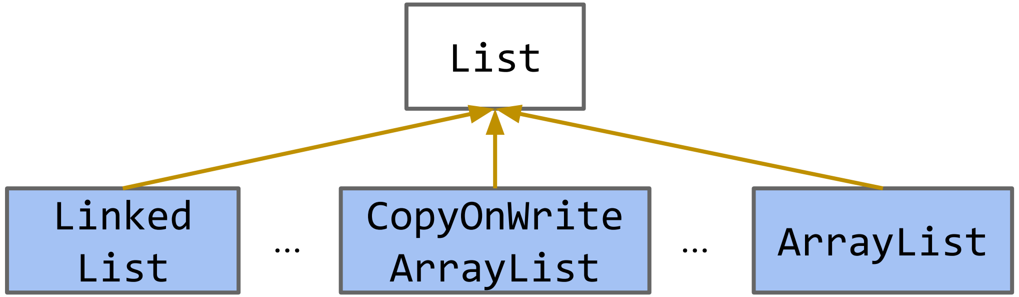


Abstract Data Types

Why bother having multiple concrete implementations of an abstract data type?

- May have better performance in certain cases. Example:
  - LinkedLists are very fast at removing the front item. ArrayLists are very slow to remove the front item.
- May have additional operations. Example:
  - The Stack implementation of List has a “push” and “pop” operation.

We’ll come to explore this concept in much more detail next week. For now, let’s get back to modernizing our Java code.



Modern Java Lists

Lecture 2, CS61B, Spring 2023

Retrieving from Lists

In Python, we retrieve items by using bracket notation.

```
L = []
L.append("a")
L.append("b")
print(L[0])
```

```
a
```

```
import java.util.List;
import java.util.ArrayList;

public class ListDemo {
    public static void main(String[] args) {
        List L = new ArrayList();
        L.add("a");
        L.add("b");
    }
}
```

Retrieving from Lists

In Java, we retrieve items using .get.

```python
L = []
L.append("a")
L.append("b")
print(L[0])
```

```
a
```

```java
import java.util.List;
import java.util.ArrayList;

public class ListDemo {
    public static void main(String[] args) {
        List L = new ArrayList();
        L.add("a");
        L.add("b");
        System.out.println(L.get(0));
    }
}
```

Java 4.0 List Limitation

The list we created is a Java 4.0 style list (from Java before 2005).

- Serious limitation: Can't (easily) assign the results of retrieving an item from a list to a variable.
- I won't teach you the Java 4.0 way to deal with this, because it is obsolete.

```
L = []
L.append("a")
L.append("b")
x = L[0]
```

Won't compile. Results in a "Required type: String, Provided: Object" error.

```
import java.util.List;
import java.util.ArrayList;

public class ListDemo {
    public static void main(String[] args) {
        List L = new ArrayList();
        L.add("a");
        L.add("b");
        String x = L.get(0);
    }
```

Java 5.0 Lists

In 2005, a new angle bracket syntax was introduced to deal with this issue.

- Instead of List L, the programmer specifies the specific type of list with List<String> L.
- Requires <> on the instantiation side.

```
L = []
L.append("a")
L.append("b")
x = L[0]
```

```
import java.util.List;
import java.util.ArrayList;

public class ListDemo {
    public static void main(String[] args) {
        List<String> L = new ArrayList<>();
        L.add("a");
        L.add("b");
        String x = L.get(0);
    }
}
```

Historical note: You had to have <String> on both declaration and instantiation sides until 2011.

Java 5.0 Lists

All items in a list created using bracket notation must have the same type!

- Unlike Python lists which can store multiple types.

```
L = []
L.append("a")
L.append(3)
# fine
```

```
import java.util.List;
import java.util.ArrayList;

public class ListDemo {
    public static void main(String[] args) {
        List<String> L = new ArrayList<>();
        L.add("a");
        L.add(3);
    }
}
```

Syntax error. Won't compile.

Limitations

Going from Java 4.0 to Java 5.0 lists meant losing the ability to have lists with multiple types.

Why might this be a good thing?

-

Limitations (My Answer)

Going from Java 4.0 to Java 5.0 lists meant losing the ability to have lists with multiple types.

Why might this be a good thing?

- It restricts the set of choices you have to make as a programmer.

Placing limitations on yourself as a programmer is a good thing!

- Freedom leads to complexity.
- Complexity is hard to fit in your brain.
- Java has MANY features that help you restrict yourself.
- Will be a recurring theme in this class.

Arrays

Lecture 2, CS61B, Spring 2023

Arrays

Java has a special collection called an “array” that is a restricted version of the list ADT.

- Size that must be declared at the time the array is created.
- Size cannot change.
- All items must be of the same type.
- No methods.
- Syntax for accessing array entries is similar to Python, e.g. x[0].

No python equivalent.

```
public class ArrayDemo {
    public static void main(String[] args) {
        String[] x = new String[5]; // size 5
        x[0] = "a";
        x[1] = "b";
        System.out.println(x[0]);
    }
}
```

Question for You

Why do you think Java has both lists and arrays (basically lists with fewer features)?

Question for You (My Answers)

Why do you think Java has both lists and arrays?

- Arrays are more performant:
  - Reading and writing from them is faster.
  - Use less memory.
- You’ll learn more about this is 61C.

Question for You

Why do you think Java favors arrays over lists (has special bracket syntax, doesn’t require imports)?

Question for You (My Answer)

Why do you think Java favors arrays over lists (has special bracket syntax, doesn't require imports)?

- Java is a language built for performance.
- By contrast, Python is built to be beautiful/simple/elegant.

Side note, when you take 61C, you'll learn C.

- C is more challenging to write than Java.
- C is more oriented around performance.

Maps

Lecture 2, CS61B, Spring 2023

Maps

In programming languages, a map is collection of key-value pairs. Each key is guaranteed to be unique. Also called:

- Dictionary (in Python).
- Associative Array (in theoretical computer science).
- Symbol Table (in Princeton’s CS61B equivalent course, not sure where else).

Example: {“alpha”: “first letter of greek alphabet”,
“potato”: “a starchy edible plant”}

For more, see: https://en.wikipedia.org/wiki/Associative_array

Maps in Python

Below, we see a simple example where we create a dictionary (a.k.a. map) in Python.

- Let’s try to do the same thing in IntelliJ.

```
m = {}
m["cat"] = "meow"
m["dog"] = "woof"
sound = m["cat"]
```

Maps in Python

On HW0B (due Monday) you'll get more practice with maps.

```python
m = {}
m["cat"] = "meow"
m["dog"] = "woof"
sound = m["cat"]
```

```java
import java.util.Map;
import java.util.TreeMap;

public class MapDemo {
  public static void main(String[] args) {
    Map<String, String> L = new TreeMap<>();
    L.put("dog", "woof");
    L.put("cat", "meow");
    String sound = L.get("cat");
  }
}
```

Summary

Lecture 2, CS61B, Spring 2023

Most Common List and Map Implementations

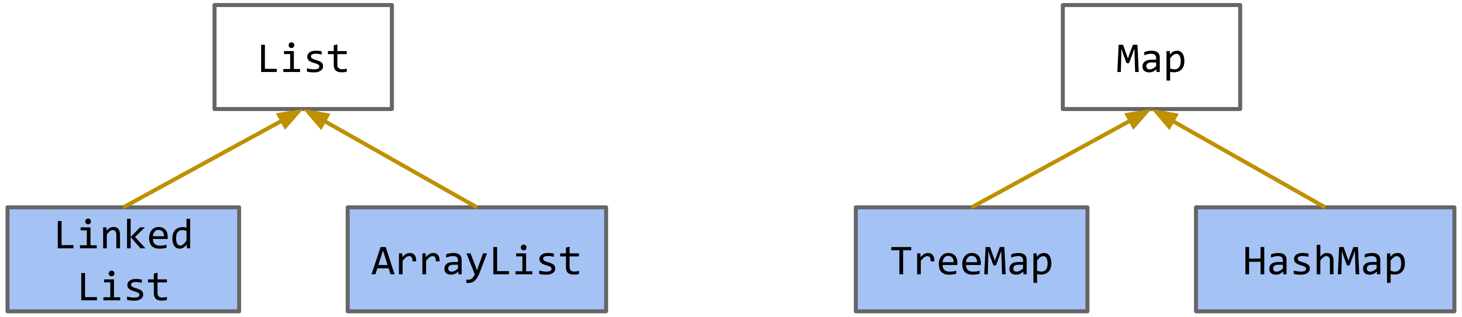
The most common Map implementations are TreeMap and HashMap.

- Python uses an ArrayList (without telling you).
- Python uses a HashMap (without telling you).

Starting next week, we'll be digging into the distinction between LinkedLists and ArrayLists.

- Will take about 3 lectures to understand.

Will more fully discuss TreeMaps and HashMaps in roughly week 7 of the course.



Todo!

Now it’s time to get started on HW0B.

- Good news: It’s short like HW0A!

Project 0 will be released by tomorrow evening.

- Feel free to get started as soon as you finish HW0B.

Don’t forget about the week 1 to 5 checklist.

- https://docs.google.com/document/d/1lzEGT8g4SmNQsfoXeSxj6kja0MhYr6x1-IH5WDtxPMQ/edit?tab=t.0

And if you just joined the class, make sure to see https://sp25.datastructur.es/policies/#late-adding-cs61b