Object Oriented Programming and Design in Java

Session 2

Instructor: Bert Huang

Announcements

- TA: Yipeng Huang, yh2315, Mon 4-6
- OH on MICE clarification
- Next Monday's class canceled for Distinguished Lecture: Feb 1, 11 AM Davis Auditorium.
- Course survey due by next class for 1 point extra credit

Schedule

Sun	Mon	Tue	Wed	Thu	Fri
John 1-3	Class 11-12:15 Bert 2-4 Yipeng 4-6	Lauren 5:30-7:30	Class 11-12:15		

Three Rivers in Machine Learning: Data, Computation and Risk

John Lafferty
Carnegie Mellon University

Abstract:

Machine learning is a confluence of computer science and statistics that is empowering technologies such as search engines, robotics, and personalized medicine. Fundamentally, the goal of machine learning is to develop computer programs that predict well, according to some measure of risk or accuracy. The predictions should get better as more historical data become available. The field is developing interesting and useful frameworks for building such programs, which often demand large computational resources. Theoretical analyses are also being advanced to help understand the tradeoffs between computation, data, and risk that are inherent in statistical learning. Two types of results have been studied: the consistency and scaling behavior of specific convex optimization procedures, which have polynomial computational efficiency, and lower bounds on any statistically efficient procedure, without regard to computational cost. This talk will give a survey of some of these developments, with a focus on structured learning problems for graphs and shared learning tasks in high dimensions.

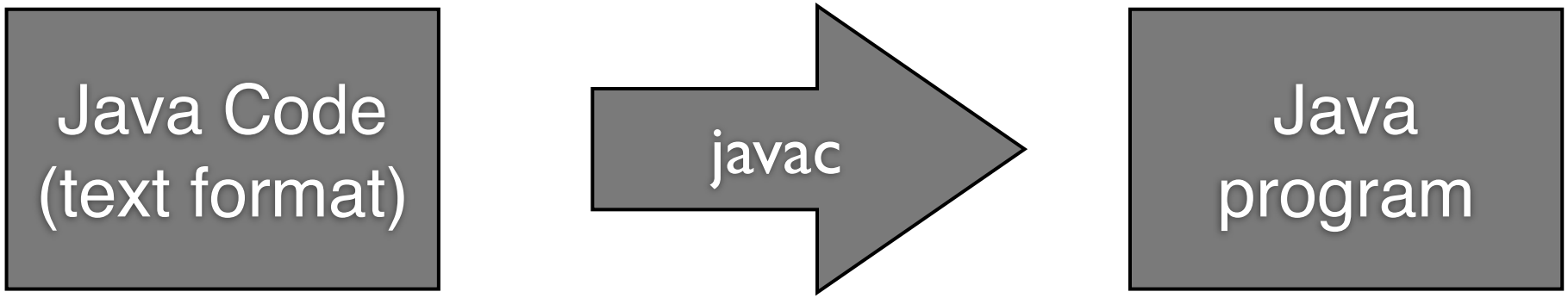
Review

- Course information
  - Prerequisites
  - Assignments and expectations
- Course goals

Today's Plan: Java Review

- basic syntax, javadoc, primitive types, references, importing packages, exceptions, input, Arrays, ArrayLists, declaration keywords, code style
- CUNIX and Eclipse demo

Compiling Java



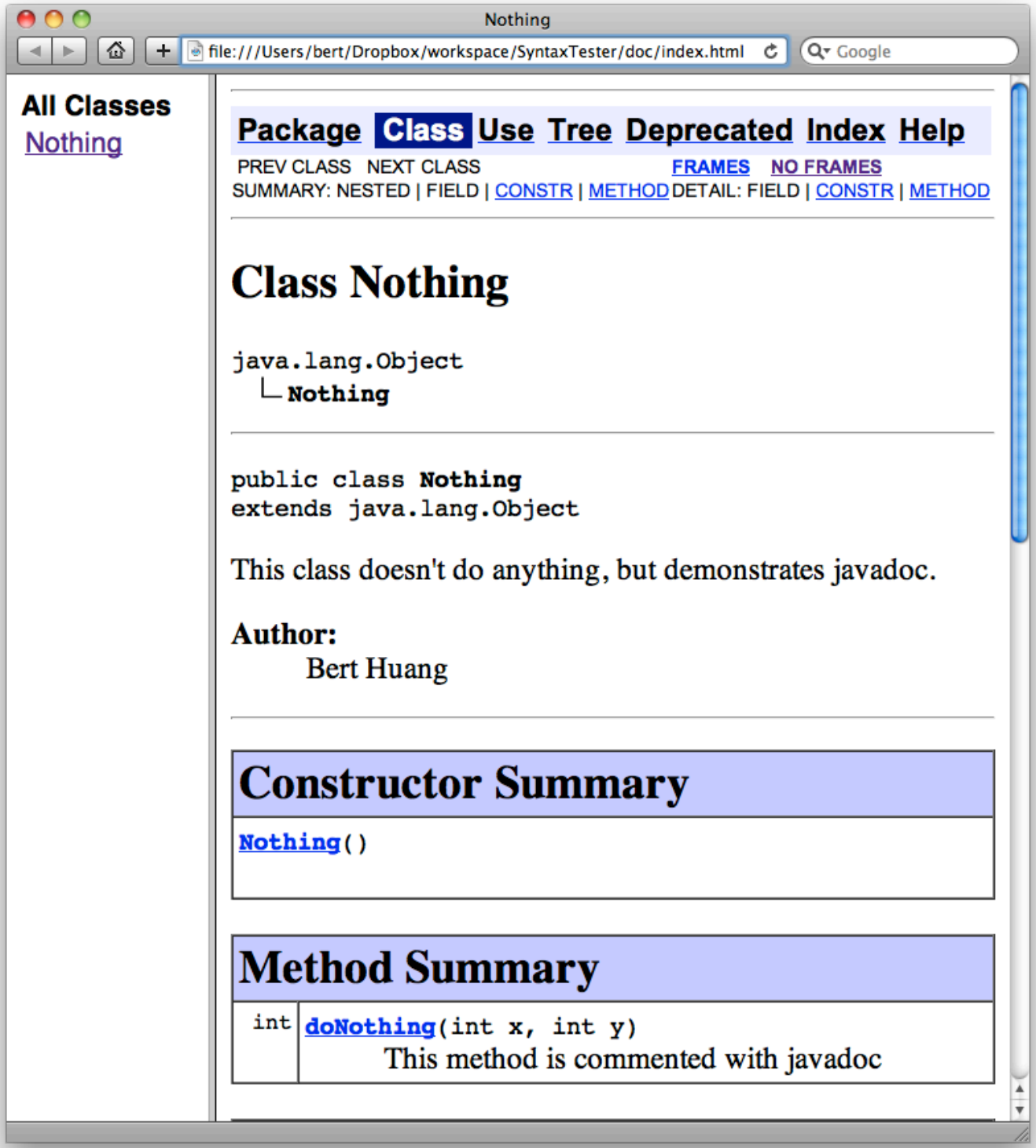
- Java is a compiled language
- Your code tells the compiler what machine code to produce

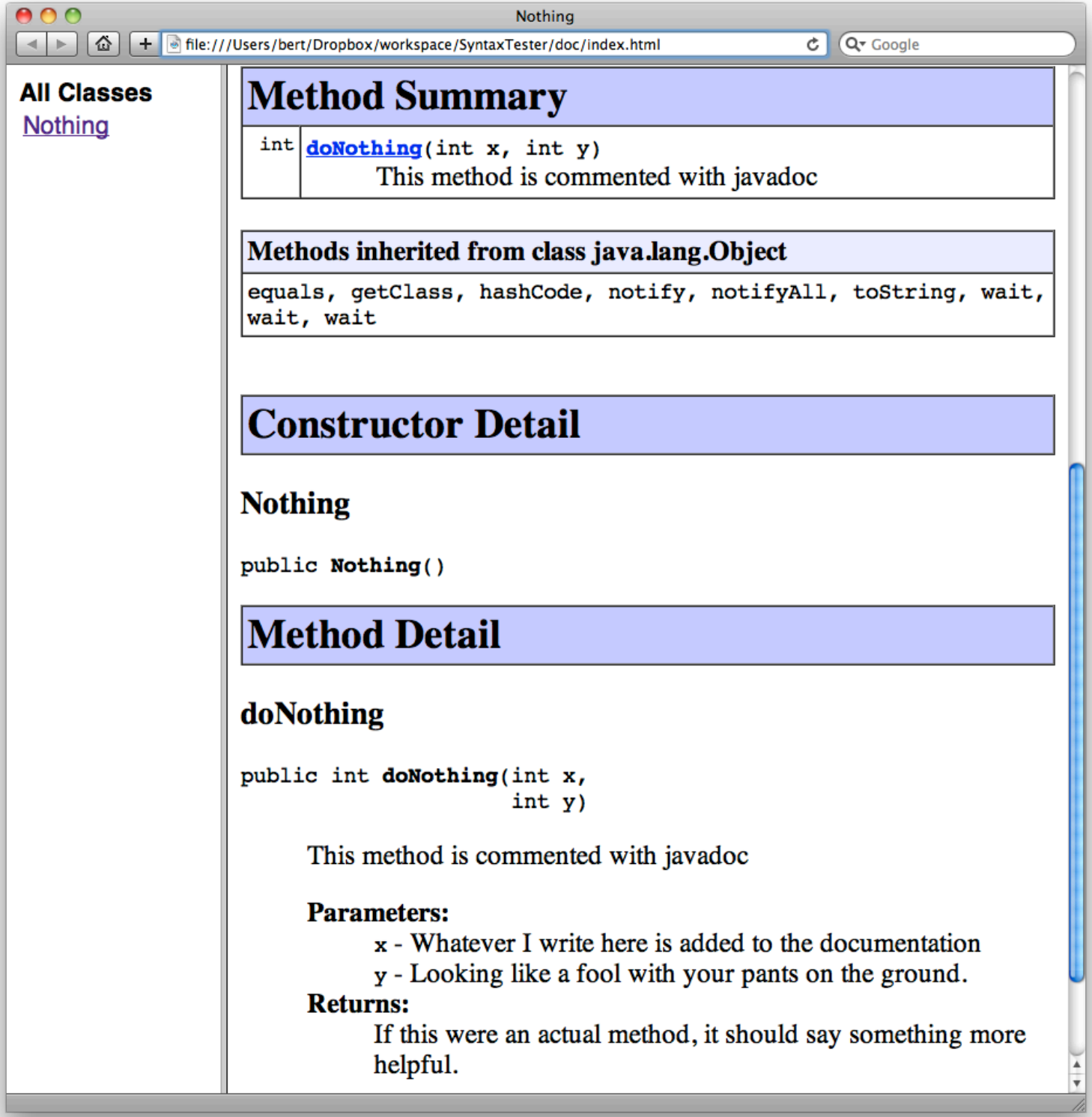
```
public class SyntaxTester {
   public static void main(String [] args)
   {
      for (int i = 0; i < MAX; i++)
      {
         System.out.println("Iteration " + i);
         if (i == 1)
         {
            System.out.println("i is 1");
         }
         else
         {
            System.out.println("i is not 1");
         }
      }
   }
   final static int MAX = 5;
}
```

Javadoc

- Automated documentation generation
- Comment each class, each public method, its parameters and return value
- javadoc converts comments into organized html website

```java
/**
 * This class doesn't do anything,
 * but demonstrates javadoc.
 * @author Bert Huang
 */
public class Nothing {
   /**
    * This method is commented with javadoc
    * @param x Whatever I write here is
    * added to the documentation
    * @param y Looking like a fool with
    * your pants on the ground.
    * @return If this were an actual method,
    * this should say something more helpful.
    */
   public int doNothing(int x, int y)
   {
      return 1000;
   }
}
```


Nothing
file:///Users/bert/Dropbox/workspace/SyntaxTester/doc/index.html
Google
All Classes
Nothing
Package Class Use Tree Deprecated Index Help
PREV CLASS NEXT CLASS
FRAMES NO FRAMES
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD
Class Nothing
java.lang.Object
Nothing
public class Nothing
extends java.lang.Object
This class doesn't do anything, but demonstrates javadoc.
Author:
Bert Huang
Constructor Summary
Nothing()
Method Summary
int
doNothing(int x, int y)
This method is commented with javadoc

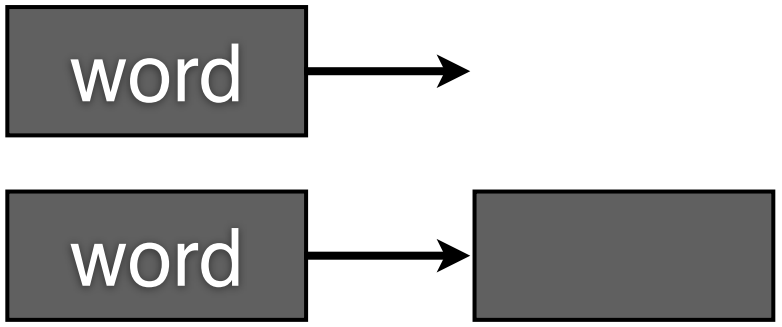

Nothing
file:///Users/bert/Dropbox/workspace/SyntaxTester/doc/index.html
Google
All Classes
Nothing
Method Summary
int
doNothing(int x, int y)
This method is commented with javadoc
Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait
Constructor Detail
Nothing
public Nothing()
Method Detail
doNothing
public int doNothing(int x, int y)
This method is commented with javadoc
Parameters:
x - Whatever I write here is added to the documentation
y - Looking like a fool with your pants on the ground.
Returns:
If this were an actual method, it should say something more helpful.

Primitive Types

- **int**, long, short, byte
- **char**
- **boolean**
- **double**, float
- Java allocates memory for each variable of these types

Object References

- All other variables are references to **objects**
- Running a constructor allocates memory for the variable
- String word;
- word = new String();

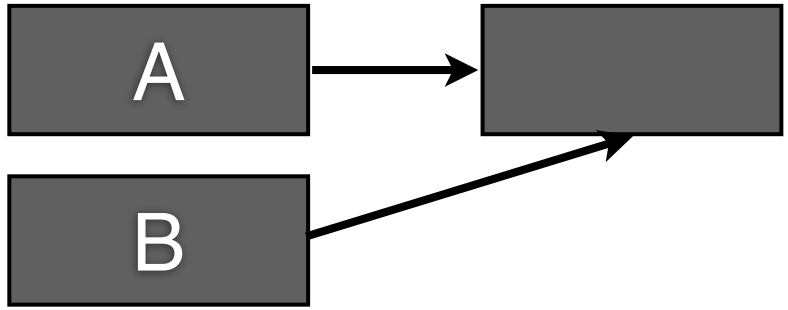


Primitive Type Wrappers

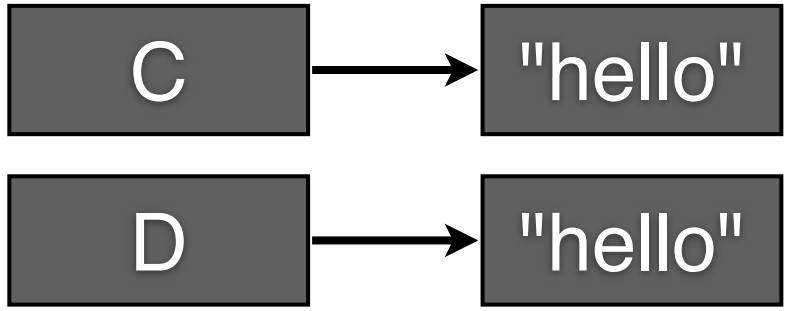
- You can use primitive types as objects by using wrappers:
- Integer, Byte, Short, Long
- Double, Float
- Character
- Boolean

Reference Headaches to Avoid

- MyObject A = new MyObject();
  MyObject B = A;



- String C = "hello";
  String D = "hello";
  C == D?



  - Use foo.equals(bar) instead

Scope

```
i = 5;
for (int i = 0; i < 10; i++)
{
    System.out.println(i);
}

System.out.println(i);
```

Scope

```
public class Example
{
    public Example(String name)
    {
        System.out.println(name);
        this.name = name;
    }
    private String name;
}
```

Importing Packages

- To import built in (or 3rd party) packages: import java.util.Random;
- Adds classes in package to your space
- You can always refer to classes by their full name:

```
java.util.Random rand = new java.util.Random();
```

Scanners for Input

- `import java.util.Scanner;`
- `Scanner in = new Scanner(System.in);`
- ```
  in.nextLine();          // returns String until \n
  in.nextInt();           // reads next token as int
  in.nextDouble();        //                    double
  in.useDelimiter(",");   // sets token separator to ,
  in.next();              // returns next token
  ```
- `new Scanner(new File("fileToRead.txt"));`
  ```

Handling Exceptions

- Methods can throw **Exceptions** to indicate runtime errors
- Methods are declared to throw exceptions, and javac complains if you don't handle them

- try
```
{
    // something
    // dangerous
}
catch (Exception e)
{
    // handle e
}
```

Throwing Exceptions

- ```
  public void myMethod() throws IOException
  {
      // do something that might throw IOException

  }
  ```
- Transfers responsibility to caller of myMethod()
- Try to handle exception asap
  ```

Arrays

- Objects can be grouped into arrays of similar objects
- int [] A = new int[5];

0	1	2	3	4
A[0]	A[1]	A[2]	A[3]	A[4]

- A[5] causes `IndexOutOfBoundsException`

ArrayLists

- `ArrayList<String> list = new ArrayList<String>();`
- No fixed size, grows as needed
- Class in < > indicates what is stored in the ArrayList
- ```
  list.add("blah");
  list.add("blah", 4);
  ```
  ```

Enhanced For Loop

- For each element of array or ArrayList
- ```
  Arraylist<String> list;

  //... put stuff into list ...

  for (String current : list)
  {
      // do something with current String
  }
  ```
  ```

Declaration Keywords

- public - Available to all other objects
- private - Available only to this object
- static - Exists independently of instantiation. Behaves the same for each instance of the class.
- final - Will never change. Use this for constants

Code Style

- Consistency is most important; easily find code
- Some good suggestions by Horstmann:
  http://www.horstmann.com/bigj/style.html
- Some highlights:
  - lowercase variable and methods
  - class names Uppercase
  - No magic numbers, use ALL_CAPS final constants
  - Put spaces around all binary operations (==, +, >)
  - Space after every comma

CUNIX and Eclipse

Reading

- Horstmann Ch. 2 for next two sessions
- Cunix info page http://www.cs.columbia.edu/~bert/courses/1007/cunix.html