Penn Engineering
Online Learning

Video 1.1

Arvind Bhusnurmath

Java

- Designed to serve the need for portability in the 90s.
- Java Virtual Machine
- Static typed
- Object oriented
- Vast collection of library packages
- One of the most popular languages
  - TIOBE index (www.tiobe.com)

Install Java

- Search for "java jdk"
- Find your operating system
- Download and install

Development Environment

How to choose a development environment

- IDE (integrated development environment)
- Eclipse
  https://eclipse.org/

eclipseinstaller by Oomph

Eclipse IDE for Java Developers

The essential tools for any Java developer, including a Java IDE, a Git client, XML Editor, Mylyn, Maven and Gradle integration.

Installation Folder /Users/bhusnur/eclipse/java-neon

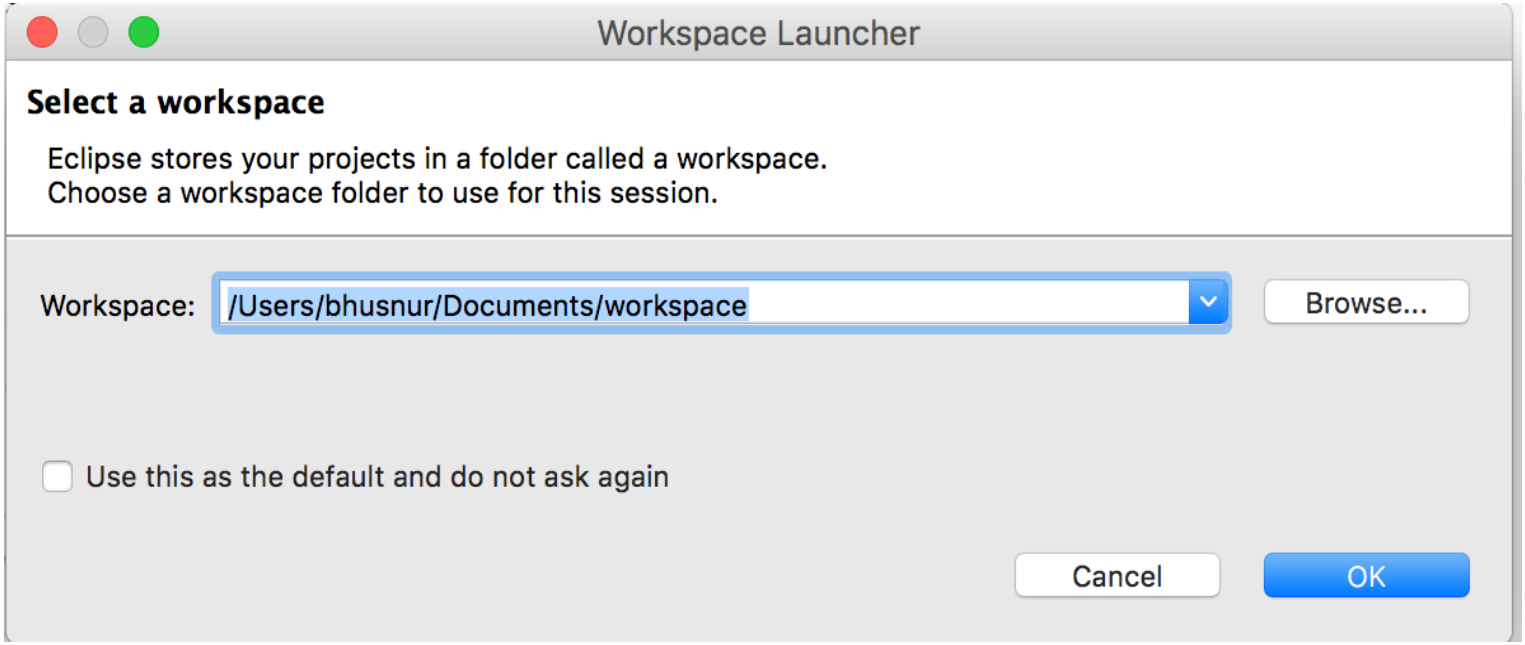
INSTALLING

Cancel Installation

BACK

Eclipse

- Go to Eclipse.org and install the latest version of Eclipse
- Install "Eclipse IDE for Java Developers"
- Launch Eclipse



workspace - Java - Eclipse

File Edit Navigate Search Project Run Window Help

Quick Access

Package Explorer

- Basic Java
  - src
    - basics
      - HelloWorld.java
  - JRE System Library [JavaSE-1.8]

Task List

Find All Activate...

Outline

An outline is not available.

Problems Javadoc Declaration

0 items

Description	Resource	Path	Location	Type

Ask me anything

10:47 AM
1/12/2017

Penn Engineering
Online Learning

Video 1.2

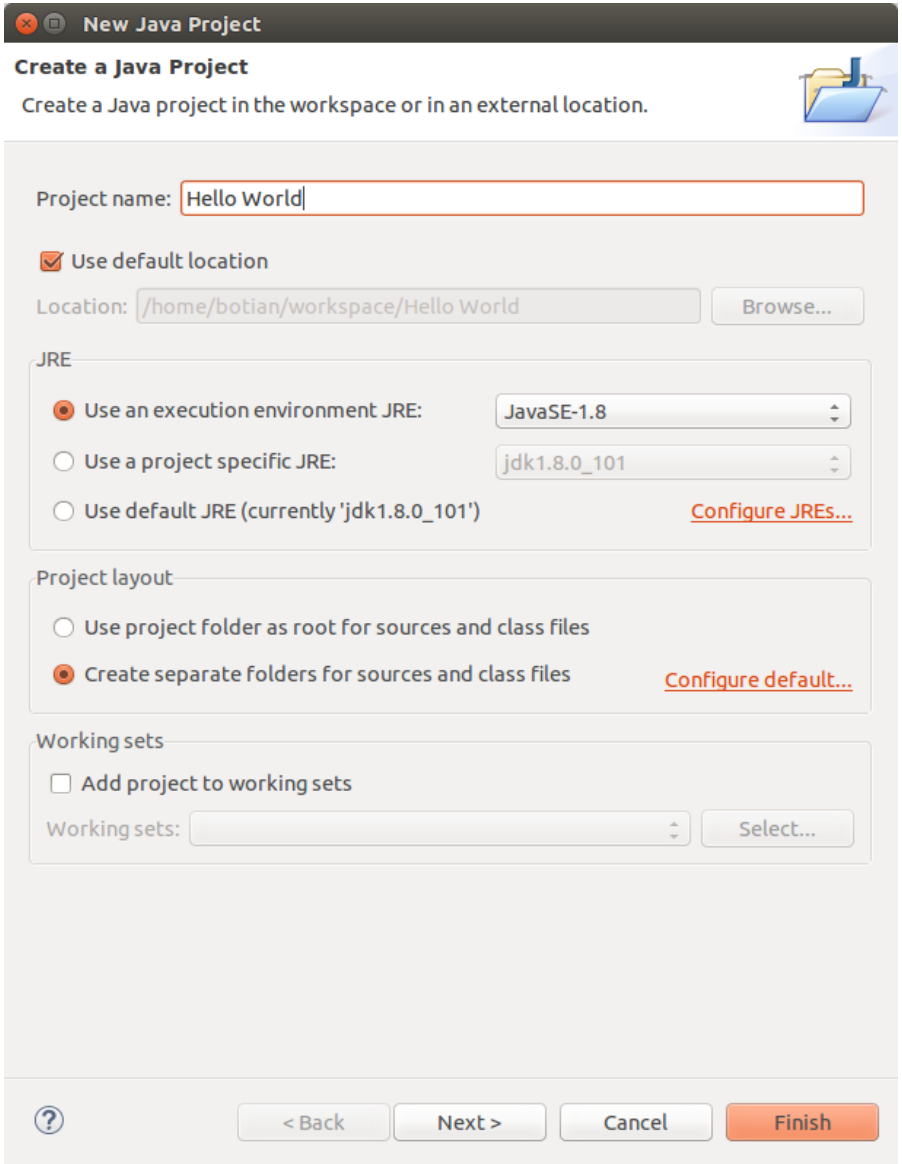
Arvind Bhusnurmath

Hello World!

- A workspace is where Eclipse keeps projects.
- When you use Eclipse to create a project, it creates a folder(directory) with that name in your workspace.
- Create a package and create a class in that package.
- For the simplest program, you need only a single package, and only one (or a very few) classes.

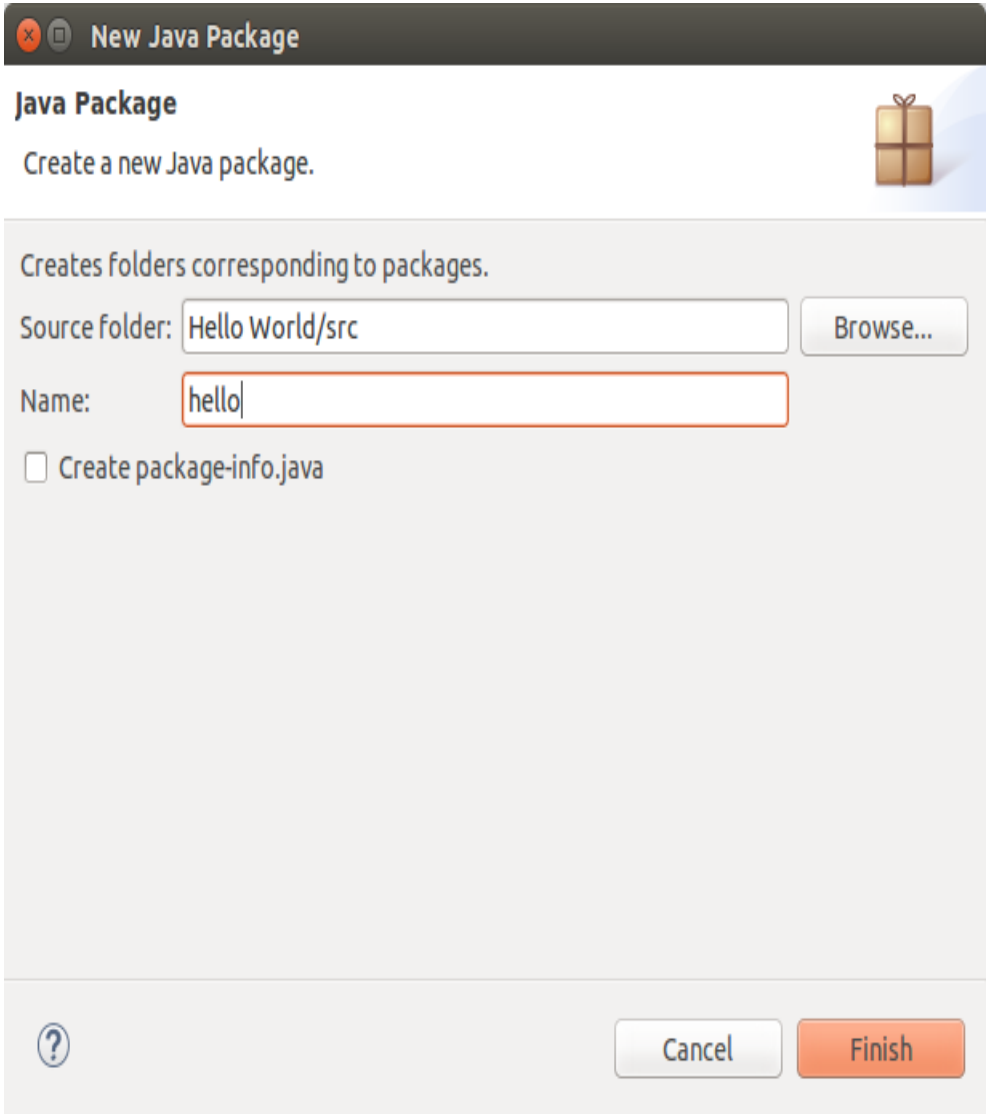
File → New → Java Project

Enter project name, and click Finish



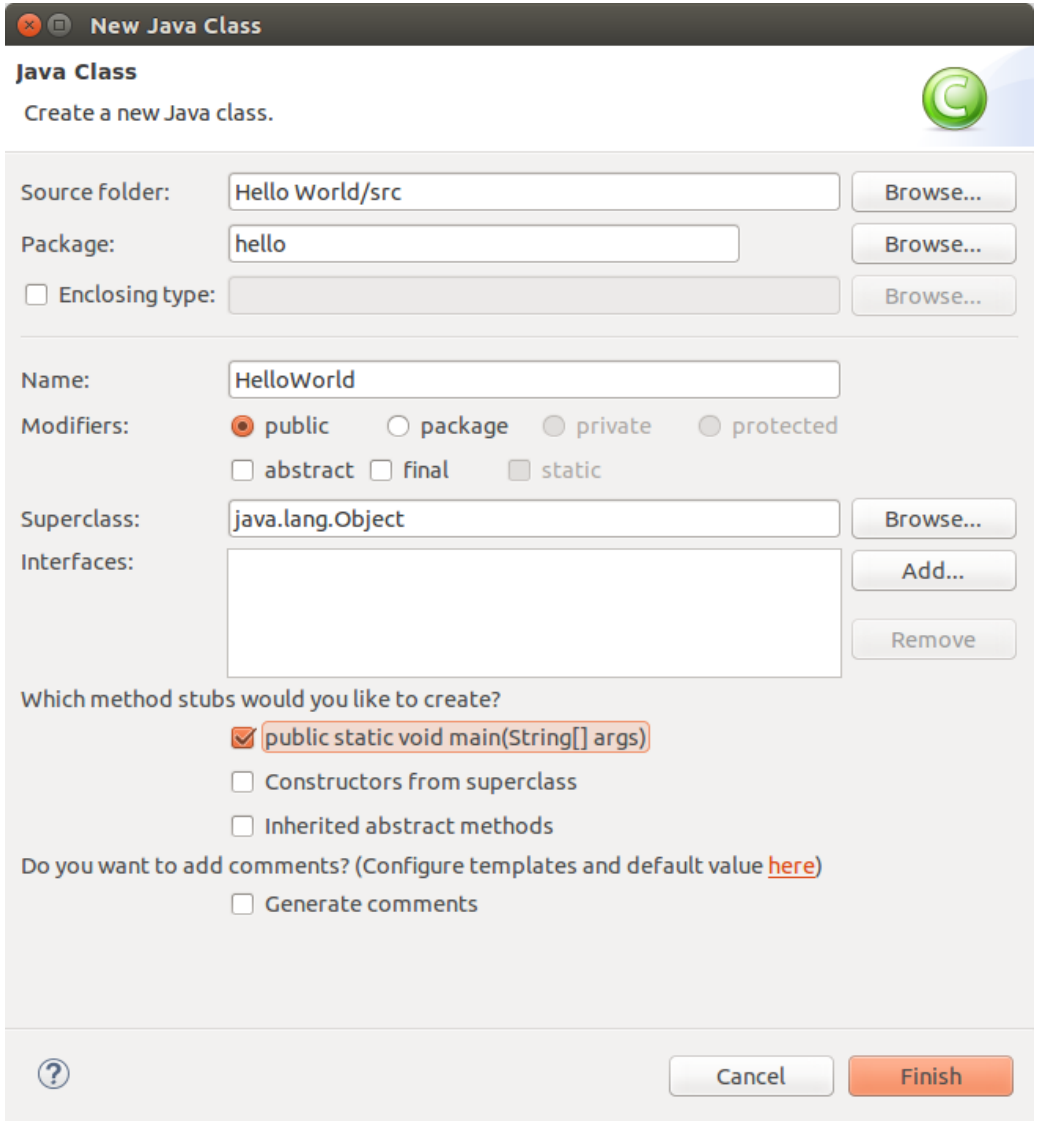
Right click on project → New → Package

Enter package name, and click Finish



Right click on package → New → Class

Enter class name, choose public static void main(String[] args), and click Finish



Quick Access | Java | Debug

Package Explorer | JUnit

- Friends-testMe
- Hello World
  - src
    - hello
      - HelloWorld.java
      - Single.java
  - JRE System Library [JavaSE-1.8]
- Meeting Optimizer
- Meeting Scheduler
- PiazzaCrawler
- Predictive Text
- RecoveryBlock-Graphs
- Test
- Test-me
- ThreadDemo
- TicTacToe
- Tweets-imSlow
- Whac-A-Mole

Dummy.java | Single.java | HelloWorld.java

```
1  package hello;
2
3  public class HelloWorld {
4
5      public static void main(String[] args) {
6          System.out.println("Hi there!");
7      }
8
9  }
10
```

Task List

Find | All | Activate...

Connect Mylyn

Connect to your task and ALM tools or create a local task.

Outline

- hello
- HelloWorld
  - main(String[]) : void

Problems | Javadoc | Declaration | Console | Debug | Variables | Breakpoints | Coverage

<terminated> HelloWorld [Java Application] /usr/lib/jvm/jdk1.8.0_101/bin/java (Feb 4, 2017, 11:05:56 AM)

Writable | Smart Insert | 6 : 41

```
public class HelloWorld{
      public static void
main(String[] args){

System.out.println("Hello World!");
      }
}
```

Run the program by clicking "Run" in the menu.

Variables and Types

Declaring a variable

```
double distance;
String firstName;
```

Declaring and initializing a variable

```
int count = 0;
```

Use/update a variable that you declared previously

```
distance = 5.67;
```

Numeric Data

- int for integers
- double if you have data with a decimal point.
- `double pi = 3.1415;`

Strings and Characters

- `char ch = 'a';`
- `String name = "Superman";`
- Difference between single characters and a string of characters.

Booleans

- Variables that either take the value **true** or **false**
- And operation - &&
- Or operation - ||
- not operation - !

Conditionals

```
if (x < 5) {
    System.out.println("less than
5");
}
else {
    System.out.println("more than
5");
}
```

If the condition is satisfied do everything within the first set of braces, otherwise execute statements within the second set of braces.

Nested Conditionals

```
if (condition1) {
    //code block 1
    if (condition2) {
        //code block 2
    }
}
```

code block 2 only gets executed if condition1 and condition2 are both satisfied.

Penn Engineering
Online Learning

Video 1.3

Arvind Bhusnurmath

Loops

Loop = doing something over and over

- Problem – Sum of the numbers from 1 to 100 = ?
- int sum = 1 + 2 + 3 + 4 + 5 + ….. + 100
- Adding repeatedly

While loops

```
int sum = 0;

int number = 1;

while (number <= 100) {
    sum = sum + number;
    number = number + 1;
}
```

The for loop

- The **for** loop is complicated, but *very* handy
- Syntax:

  for (***initialize*** ; ***test*** ; ***increment***) ***statement*** ;

  Notice that there is no semicolon after the ***increment***
- Execution:

  The ***initialize*** part is done first and only once

  The ***test*** is performed; as long as it is true,
  - The ***statement*** is executed
  - The ***increment*** is executed

Sum of numbers using a for loop

```
int sum = 0;
for (int number = 1; number <= 100;
number = number + 1) {
    sum = sum + number;
}
System.out.println(sum);
```

Sum of numbers using a for loop

```
int sum = 0;
for (int number = 1; number <= 100;
number = number + 1) {
    sum = sum + number;
}
System.out.println(sum);
```

Sum of numbers using a for loop

```
int sum = 0;
for (int number = 1; number <= 100;
number = number + 1) {
    sum = sum + number;
}
System.out.println(sum);
```

Sum of numbers using a for loop

```
int sum = 0;
for (int number = 1; number <= 100;
number = number + 1) {
    sum = sum + number;
}
System.out.println(sum);
```

When do you use each loop

- Use the for loop if you know ahead of time how many times you want to go through the loop
  - Example: Stepping through an array
  - Example: Print a 12-month calendar
- Use the while loop in almost all other cases
  - Example: Compute the next step in an approximation until you get close enough

Scope of a variable

- Block of code – statements between braces {}
- Access to a variable is limited to the block of code where it is defined.
- for (int i = 0; ……; …..) - the variable i is scoped to the contents of the loop. If you try and access i outside the loop it causes a compile error.

The increment and decrement operations

x++ is Java shorthand for x = x + 1;
x-- is Java shorthand for x = x – 1;

Extremely common to see this syntax in loops

Example: print hello 10 times

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

Penn Engineering
Online Learning

Video 1.4

Arvind Bhusnurmath

Sample Problem

Goal : Write a program that gets a single positive integer from the user and checks whether it is prime or not.

Reminder: A prime number is a number greater than 1 whose only factors are 1 and the number itself.

Reading input from the user

- First, import the Scanner class:
```
import java.util.Scanner;
```
- Create a scanner and assign it to a variable:
```
Scanner scanner = new
Scanner(System.in);
```
  - The name of our scanner is scanner
  - new Scanner(...) says to make a new one
  - System.in says the scanner is to take input from the keyboard
- Next, it's polite to tell the user what is expected:
```
System.out.print("Enter a number:  ");
```
- Finally, read in the number:
```
myNumber = scanner.nextInt();
```
- If you haven't previously declared the variable myNumber, you can do it when you read in the number:
```
int myNumber = scanner.nextInt();
```

Reading input from the user

```
import java.util.Scanner;
public class Prime {
     public static void main(String
args[]){
     Scanner scanner = new
Scanner(System.in);
     System.out.print("Enter a number:
     int myNumber = scanner.nextInt();
     }
}
```

Penn Engineering
Online Learning

Video 1.5

Arvind Bhusnurmath

Syntactic Style

Why does style matter

- Good style isn't just to make your code "look pretty"
- The most critical factor in style is readability
- If a program is readable,
  - It is easier to debug
  - It is easier to maintain
  - It is easier to upgrade
- For "real" programs (those that actually get used), the time spent *reading* them far exceeds the time spent *writing* them

Be consistent!

- Most times, you will enter an ongoing project, with established style rules
  - Follow them even if you don't like them
  - As they are what your team is used to, they will be more readable to other members of your team

Do it right the first time

- You only write code once, but you read it many times while you're trying to get it to work
  - Good style makes it more readable and *helps you get it right!*

Indent nested code

- Always indent statements that are nested inside (under the control of) another statement
  - ```
    if (itemCost <= bankBalance) {
        writeCheck(itemCost);
        bankBalance = bankBalance - itemCost;
    }
    ```
- The open brace always goes at the end of a line
- The matching close brace lines up with the statement being closed
- Don't use C-style braces unless that is the already established standard for the project you are on
- Indentation should be consistent throughout the program
  - 4 spaces is the standard for Java (other languages may differ)
    ```

Break up long lines

- Keep your lines short enough to be viewed and printed
- Many people use 72 or 80 character limits
- Suggestions on where to break a long line:
  - It's *illegal* to break a line within a quoted string
  - Break after, not before, operators
  - Line up parameters to a method
  - *Don't* indent the second line of a control statement with a long test so that it lines up with the statements being controlled

Using spaces

- Use spaces around all binary operators except "dot":

```
if (n > 1 && n % 2 == 1) n = 3 *
n + 1;
```

- Do *not* use spaces just within parentheses:

```
if ( x < 0 ) x = -x;   // don't do
this
```

- Use a space before and after the parenthesized test in a control statement:

```
if (x < 0) {...}
  while (x < 0) {...}
```

Use meaningful names

- Names should be chosen very carefully, to indicate the ***purpose*** of a variable or method
  - If the purpose changes, the name should be changed
  - *Spend a little time to choose the best name for each of your variables and methods!*
- Long, multiword names are common in Java
  - Eclipse will complete long names for you (control-space)
  - However, if a name is too long, maybe you're trying to use it for too many purposes
    - Don't change the name, separate the purposes
- Don't abbreviate names
  - But very common abbreviations, such as max for "maximum", are OK

Meaningful names: exception

- It is common practice to use i as the index of a for-loop, j as the index of an inner loop, and k as the index of a third-level loop
- This is almost always better than trying to come up with a meaningful name
- Example:

```
for (int i = 1; i <= 10; i++) {
    for (int j = 1, j <= 10; j++) {
        System.out.println("   " + (i *
j));
    }
}
```

Naming variables

- Capitalize the first letter of each word *except* the first:

  `total, maxValue`

- Use nouns to name variables:

  balance, outputLine

  - Variables are supposed to represent *values*

Naming constants

- A constant is an identifier whose value, once given, cannot be changed
- Constants are written with the keyword final, for example:
  - `final int FIVE = 5;`
  - `final float AVOGADROS_NUMBER = 6.022E23;`
- Constants are written in ALL_CAPITALS, with underscores between words

Comments

- Single-line comments start with //
- Multi-line comment start with /* and end with */
- Documentation comments start with /** and end with */, and are put just *before* the definition of a variable, method, or class

Correct (syntactic) style made easy

- In Eclipse, go to Window → Preferences → Java → Code Style → Formatter, and under Select a profile: choose Java conventions [built-in]

- Select some or all of your code and choose Source → Format

- To simply indent correctly, without reformatting, select some lines and choose Source → Correct Indentation

Penn Engineering
Online Learning

Video 1.6

Arvind Bhusnurmath

Topics

- What is a class?
- Instance variables
- Methods
- Using a class

Example

Model a parking lot with spaces for cars

Java does not have a Car datatype

Java does not have a Parking Lot datatype

Classes allow you to create your own datatype.

To model a problem in the object oriented programming world, the nouns in the problem will typically be converted into classes.

Creating a Car class

```
public class Car {

}
```

This code should be in a file called Car.java

Creating a Car class (instance variables)

- Instance variables are basically the properties of an "instance" of the class in question.
- What properties does a car have?
- Example - make, model, year of manufacture, is it new or used?, miles, owner.
- "A 2014 second hand Audi A4"

Creating a Car class

```
public class Car {
    //instance variables
    String make;
    String model;
    int year;
    boolean isNew;
    double miles; //miles the car has
traveled
    String owner;
}
```

Creating an instance of a Car

```
public class Car {
   //instance variables
   String make;
   String model;
   int year;
   boolean isNew;
   double miles; //miles the car has
traveled
   String owner;
}
public static void main(String[] args) {
   Car myCar = new Car();
}
```

The "." operator

```
myCar.make = "Audi" ;

myCar.model = "A4";

myCar.year = 2014;

myCar.miles = 0;

myCar.isNew = true;

myCar.owner = "Arvind";
```

Doing something with a car

- Sell the car
- The nouns in a problem get turned into classes
- The verbs in a problem get turned into methods
- Methods are similar to functions that you might have seen in other languages.

```
    /**
        sell the car to newOwner
    */
    public void sell(String newOwner)
{
        owner = newOwner;
        if (isNew) {
            isNew = false;
        }
    }
```

```
public class Car {
    //instance
variables
    String make;
    String model;
    int year;
    boolean isNew;
    double miles;
    String owner;
```

```
public void
sell(String newOwner)
{
        owner =
newOwner;
        if (isNew) {
            isNew =
false;
        }
    }
}

public static void
main(String[] args) {
    Car myCar = new
Car();
}
```

A method with a return Statement

- We want to know whether the car is older than 10 years

```
public boolean isOld() {
     int thisYear=
Calendar.getInstance().get(Calendar.YEAR);
     if (thisYear - year > 10) {
          return true;
     }
     else {
        return false;
     }
}
```

```
public static void main(String[] args) {
    Car myCar = new Car();
    myCar.make = “Audi” ;
    myCar.model = “A4”;
    myCar.year = 2014;
    myCar.miles = 0;
    myCar.isNew = true;
    myCar.owner = “Arvind” ;
    boolean isMyCarOld = myCar.isOld();
    myCar.sell(“John Doe”);
    System.out.println(”Car owned by” +
myCar.owner);
}
```

Penn Engineering
Online Learning

Video 1.7

Arvind Bhusnurmath

Topics

- What is a constructor?
- “this” keyword
- Using inbuilt classes

Defining Constructors

- A constructor is code to create an object
  - You *can* do other work in a constructor, but you *shouldn't*
- The syntax for a constructor is:

```
ClassName(parameters)  {
        …code…
}
```

- The *ClassName* has to be the same as the class that the constructor occurs in
- The *parameters* are a comma-separated list of variable *declarations*

Example constructor 1

```
public class Person {
    String name;
    int age;
    boolean male;

    Person (String aName, boolean isMale) {
        name = aName;
        male = isMale;
    }

}
```

Using a constructor

- The **new** keyword is used to create an instance of the class
- `Classname variable = new Classname(constructor parameters)`

Example constructor 2

- Most constructors just set instance variables:

```
public class Person {
    String name;
    boolean male;

    Person (String name, boolean male) {

        this.name = name ;

        this.male = male ;
    }
```

The this keyword

- this refers to the current instance of the class, the object in question.
- It is generally used for 2 purposes
  - disambiguate between an instance variable and a local variable, especially in constructors
  - when the entire current object has to be passed to a method

Using an inbuilt class

- A useful Java class is StringBuilder which is an efficient way of dealing with large strings.
- https://docs.oracle.com/javase/8/docs/api/java/lang/StringBuilder.html
- Scroll down to the constructor summary
- Use the most suitable constructor
- `StringBuilder s1 = new StringBuilder(“my first string builder”);`

StringBuilder

- Go to the method summary in the documentation
- use the "." operator to access methods
- append
- substring

Penn Engineering
Online Learning

Video 1.8

Arvind Bhusnurmath

Topics

- Writing methods other than a constructor
- Scope
- Return statements

Defining a method

A method has the syntax:

return-type method-name(parameters) {
method-variables
code
}

Example:

```
boolean isAdult(int age) {
    int magicAge = 21;
    return age >= magicAge;
}
```

Example:

```
double average(int a, int b) {
    return (a + b) / 2.0;
}
```

Methods may have local variables

- A method may have local (method) variables
- Formal parameters are a kind of local variable

```
int add(int m, int n) {
    int sum = m + n;
    return sum;
}
```

- m, n, and sum are all local variables
  - The scope of m, n, and sum is the method
  - These variables can *only* be used in the method, *nowhere else*
  - The *names* can be re-used elsewhere, for *other* variables

Blocks (== Compound statements)

Inside a method or constructor, whenever you use braces, you are creating a *block*, or *compound statement*:

```
int absoluteValue(int n) {
    if (n < 0) {
        return -n;
    }
    else return n;
}
```

Declarations in a method

- The scope of formal parameters is the entire method
- The scope of a variable in a block starts *where you define it* and extends *to the end of the block*

```
if (x > y) {
    int larger = x;
}
else {
    int larger = y;
}
return larger;
```

Returning a result from a method

- If a method is to return a result, it must specify the *type* of the result:
  - `boolean isAdult ( …`
- You must use a return statement to exit the method with a result of the correct type:
  - `return age >= magicAge;`

Returning *no* result from a method

- The keyword void is used to indicate that a method doesn't return a value
- The return statement must not specify a value
- Example:

```
void printAge(String name, int age) {
    System.out.println(name + " is " + age + "
years old.");
    return;
}
```

- There are two ways to return from a void method:
  - Execute a return statement
  - Reach the closing brace of the method

Sending messages to objects

- We don't perform operations on objects, we "talk" to them
  - This is called *sending a message* to the object
- A message looks like this:
  - ***object.method(extra information)***
    - The ***object*** is the thing we are talking to
    - The ***method*** is a name of the action we want the object to take
    - The ***extra information*** is anything required by the method in order to do its job
- Examples:

```
g.setColor(Color.pink);
amountOfRed = Color.pink.getRed( );
```

Putting it all together

```
class Person {

    // fields
    String name;
    int age;

    // constructor
    Person(String name) {
      this.name = name;
      age = 0;
    }

// methods
 String getName() {
   return name;
 }

  void birthday() {
    age = age + 1;
    System.out.println(
       "Happy birthday!");
  }
}
```

Using our new class

```
Person john;
john = new Person("John Smith");

System.out.print (john.getName());
System.out.println(" is having a birthday!");
john.birthday();
```

Of course, this code must *also* be inside a class!

Program structure

- A program consists of one or more classes
- Typically, each class is in a separate .java file

```
public class SomeClass {
    // instance variables

    // one or more constructors

    // methods

    // optionally a main method
}
```

null

- If you declare a variable to have a given object type, for example,

```
Person john;
String name;
```

- ...and if you have not yet assigned a value to it, for example, with

```
john = new Person();
String name = "John Smith";
```

- ...then the value of the variable is null
- null is a legal value, but there isn't much you can do with it
  - It's an error to refer to its fields, because it has none
  - It's an error to send a message to it, because it has no methods
  - The error you will see is NullPointerException

Penn Engineering
Online Learning

Video 1.9

Arvind Bhusnurmath

Topics

- Representing data collections
- Arrays
- ArrayLists

Arrays

Example: How do we keep track of the rainfall during the year?

```
double jan1 = …;
double jan2 = …;
…
```

This is tedious!

Solution : Array!

Arrays

An array is an indexed sequence of values of the same data type.

```
double[] rainfall= new double[365];
```

Notice the usage of the **new** keyword.

365 consecutive locations in memory are allocated to this array.

It is easy to index into any array location.

rainfall[0] is the very first entry in the array. 0 indexing.

Declaring arrays

rainfall

0	0	0	0	0	0	0	0	0	0

rainfall[0]

Declaring arrays

rainfall

0	0	0	0	0	0	0	0	0	0

rainfall[1]

Number of elements in an array

Get the number of elements in an array by using array.length

rainfall

0	0	0	0	0	0	0	0	0	0

rainfall[rainfall.length - 1]

Looping through an array

Assume the rainfall array is populated by the rainfall every day in inches.

```
double sum = 0;
for (int i = 0; i < rainfall.length; i++)
{
    sum += rainfall[i];
}
avgRainfall = sum / rainfall.length
```

ArrayList

- Arrays are fixed size. If you wanted to use the rainfall array to capture 2 years worth of data, it would be impossible to do so.

- ArrayLists work like arrays but with the added flexibility of methods to append and remove elements

Declaring an ArrayList

```
ArrayList<String> studentNames = new
ArrayList<String>();
```

Adding a new element

```
package basicjava;

import java.util.*;

public class CollectionExample {
    public static void main(String[] args) {
      ArrayList<String> studentNames = new
ArrayList<String>();
      studentNames.add("Student1");
      studentN ames.add("Student2");
      System.out.println(studentNames.size());
    }
}
```

ArrayList of integers

- int vs Integer
- Integer is an inbuilt java class that acts as a wrapper for the primitive datatype int.
- int, double and boolean are primitive java types.