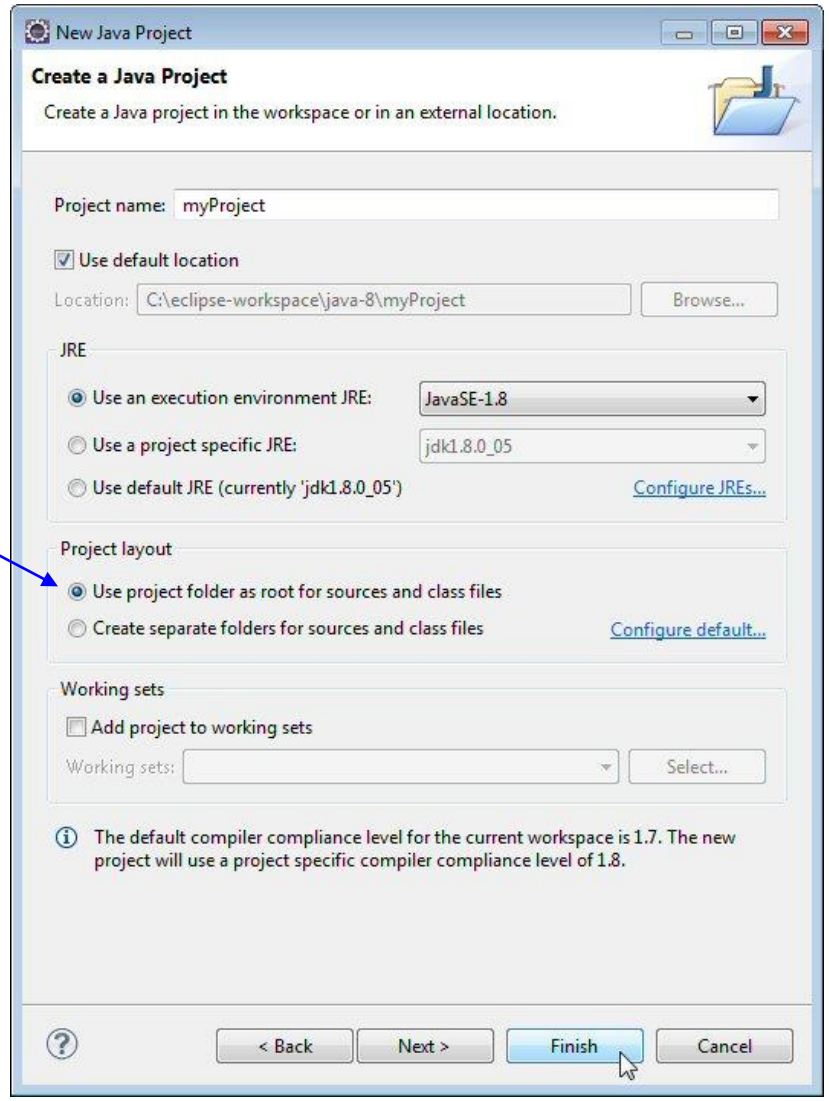
Basic Java Syntax

Topics in This Section

- Basics
- Accessing arrays
- Looping
- Indenting code
- if statements and other conditionals
- Strings
- Building arrays
- Performing basic mathematical operations
- Getting input from the user
- Converting strings to numbers

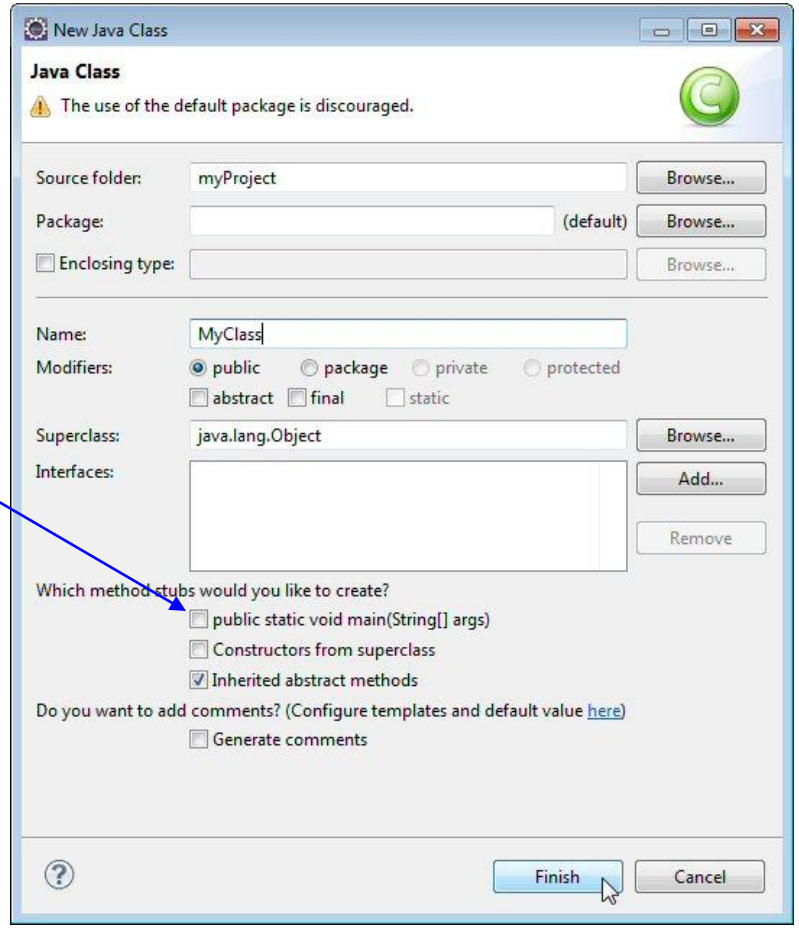
Eclipse: Making Projects

- **Main steps**
  - File → New → Project → Java → Java Project
    - Pick any name
  - If you plan to run from command line
    - Choose sources/classes in same project folder



Eclipse: Creating Classes

- **Main steps**
  - R-click on project → New → Class
  - Choose a capitalized class name (e.g., Class1 or MyFirstClass)
    - You can have Eclipse make "main" when class is created, but easier to use shortcut to insert it later
    - Eventually you will make package (subdirectory) first, then put class there Packages explained in upcoming section
- **Alternative**
  - Can also copy/rename existing class

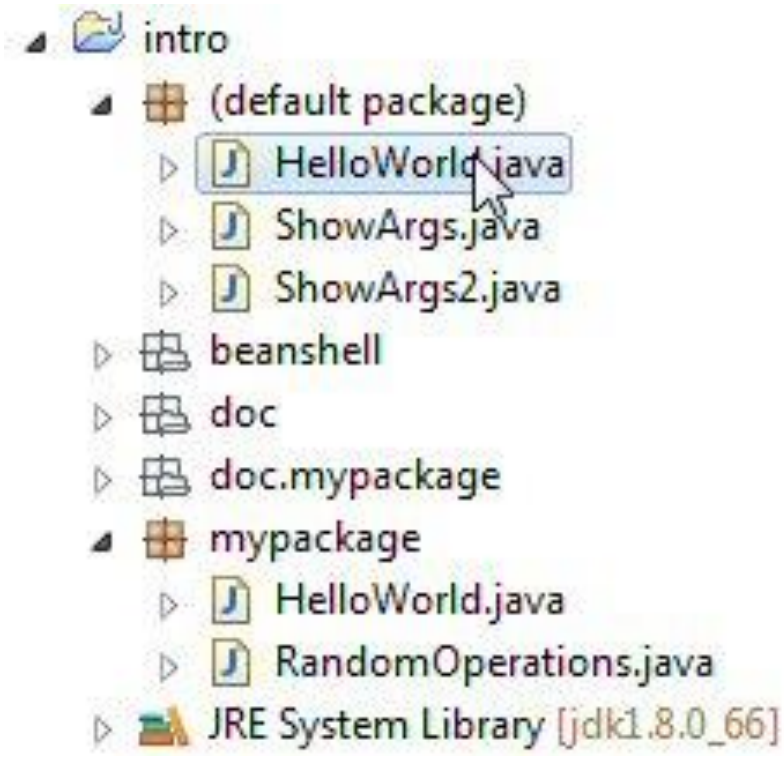


Getting Started: Syntax

- **Example**

```
public class HelloWorld {
  public static void main(String[] args) {
    System.out.println("Hello, world.");
  }
}
```

- **Details**
  - Processing starts in main
    - Eclipse can create main automatically
      - When creating class: choose main as option
      - Eclipse shortcut inside class: type “main” then hit Control-space
    - Routines usually called “methods,” not “functions.”
  - Printing is done with System.out.print...
    - System.out.println, System.out.print, System.out.printf
    - Eclipse shortcut: type “sysout” then hit Control-space



Getting Started: Execution

- **File: HelloWorld.java**

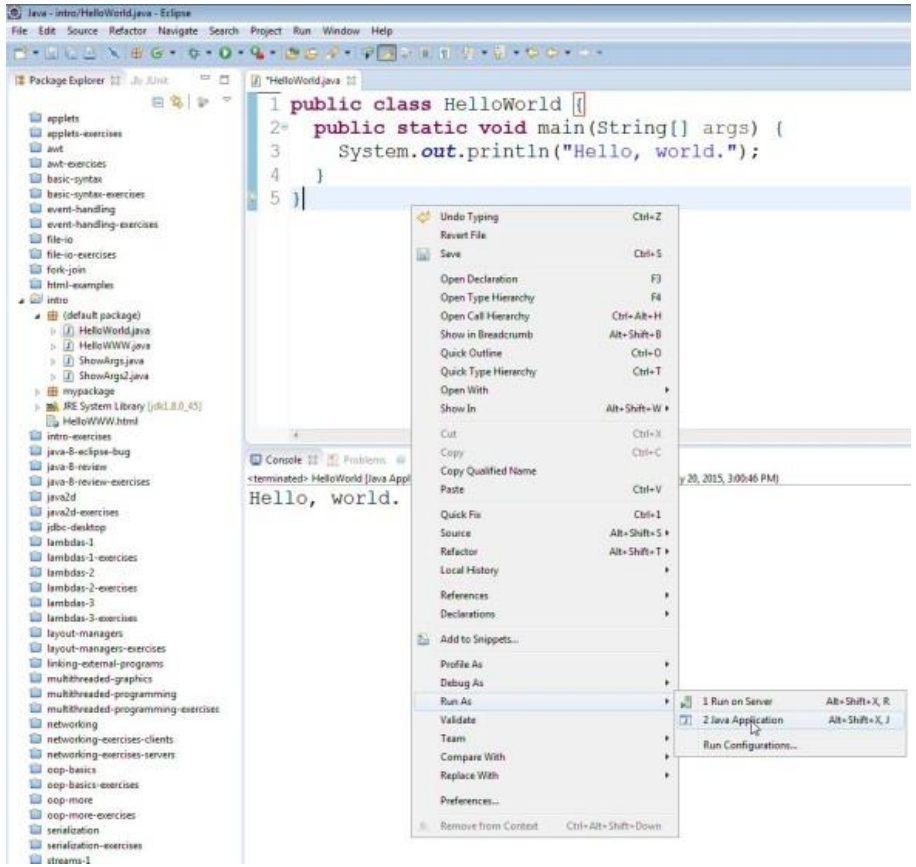
```
public class HelloWorld {
   public static void main(String[] args) {
      System.out.println("Hello, world.");
   }
}
```

- **Compiling**
  - Eclipse: just save file

```
> javac HelloWorld.java
```

- **Executing**
  - Eclipse: R-click, Run As, Java Application

```
> java HelloWorld
Hello, world.
```



Packages

- **Idea**
  - Packages are subdirectories used to avoid name conflicts
  - Java class must have "package subdirname;" at the top
    - But Eclipse puts this in automatically when you right-click on a package and use New → Class
- **Naming conventions**
  - Package names are in all lower case
  - Some organizations use highly nested names
    - com.companyname.projectname.projectcomponent
- **Creating packages in Eclipse**
  - R-click project, New → Package (use all-lowercase name by convention)
  - Then R-click package and New → Class (use capitalized name by convention)

HelloWorld with Packages (in src/mypackage folder)

```
package mypackage;

public class HelloWorld {
  public static void main(String[] args) {
    System.out.println("Hello, world (using packages)");
  }
}
```

Run from Eclipse in normal manner: R-click, Run As → Java Application. Running from the command line is a pain: you must go to *parent* directory and do "java mypackage.HelloWorld". Run from Eclipse and it is simple to use packages.

The + Operator

- **Use + for addition**
  - If *both* arguments are numbers, + means addition.
  - Example:

```
double result = 2.3 + 4.5;
```

- **Use + for string concatenation**
  - If *either* argument is String, + means concatenation
  - + is only overloaded operator (operator with multiple meanings) in all of Java
  - Examples

```
String result1 = "Hello, " + "World";  // "Hello, World"
String result2 = "Number " + 5;        // "Number 5"
```

Array Basics: Accessing Elements

- **Arrays are accessed with [ ]**
  - Array indices are zero-based

```
int[] nums = { 2, 4, 6, 8 };
```

    - nums[0] is 2
    - nums[3] is 8
    - Trying to access nums[4] results in error
- **Main is passed an array of strings**
  - args[0] returns first command-line argument
  - args[1] returns second command-line argument, etc.
  - Error if you try to access more args than were supplied

Array Basics: The length Field

- **The length variable tells you number of array elements**
  - Gives the number of elements in *any* array

```
String[] names = { "John", "Jane", "Juan" };
```

    - names.length is 3
    - But last entry ("Juan") is names[2], not names[3]
  - For command-line arguments
    - In main, **`args.length`** gives the number of command-line arguments
    - Unlike in C/C++, the name of the program is not inserted into the command-line arguments

Command-line Arguments

- **Are useful for learning and testing**
  - Command-line args are slightly helpful for beginner's practice
  - But, programs given to end users should almost never use command-line arguments
    - They should pop up a GUI to collect input
- **Eclipse has poor support**
  - Entering command-line args via Eclipse is more trouble than it is worth
  - So, to test with command-line args:
    - Save the file in Eclipse (causing it to be compiled)
    - Navigate to folder on desktop (not within Eclipse)
    - Open command window (Start icon, Run… → cmd)
    - Type "java *Classname* arg1 arg2 …"

Example: Command Line Args and length Field

- **File: ShowTwoArgs.java (naïve version)**

```
public class ShowTwoArgs {
  public static void main(String[] args) {
    System.out.println("First arg: " + args[0]);
    System.out.println("Second arg: " + args[1]);
  }
}
```

Oops! Crashes if there are fewer than two command-line arguments. The code should have checked the length field, like this:

```
if (args.length > 1) {
  doThePrintStatements();
} else {
  giveAnErrorMessage();
}
```

Example (Continued)

- **Compiling (automatic on save in Eclipse)**

```
> javac ShowTwoArgs.java
```

- **Manual execution**

```
> java ShowTwoArgs Hello Class
First arg: Hello
Second arg: Class

> java ShowTwoArgs
[Error message]
```

- **Eclipse execution (cumbersome)**
  - To assign command line args: R-click, Run As, Run Configurations, click on "Arguments" tab

Looping Constructs

- **for/each**

```
for(variable: collection) {
  body;
}
```

- **for**

```
for(init; continueTest; updateOp) {
  body;
}
```

- **while**

```
while (continueTest) {
  body;
}
```

- **do**

```
do {
  body;
} while (continueTest);
```

For/Each Loops

```
public static void listEntries(String[] entries) {
   for(String entry: entries) {
     System.out.println(entry);
   }
}
```

- **Result**

```
String[] test = {"This", "is", "a", "test"};
listEntries(test);

This
is
a
test
```

For Loops

```
public static void listNums1(int max) {
   for(int i=0; i<max; i++) {
     System.out.println("Number: " + i);
   }
}
```

- **Result**

```
listNums1(4);

Number: 0
Number: 1
Number: 2
Number: 3
```

While Loops

```
public static void listNums2(int max) {
   int i = 0;
   while (i < max) {
      System.out.println("Number: " + i);
      i++;    // "++" means "add one"
   }
}
```

- **Result**

```
listNums2(5);

Number: 0
Number: 1
Number: 2
Number: 3
Number: 4
```

Do Loops

```
public static void listNums3(int max) {
   int i = 0;
   do {
      System.out.println("Number: " + i);
      i++;
   } while (i < max);
                  // ^ Don't forget semicolon
}
```

- **Result**

```
listNums3(3);
Number: 0
Number: 1
Number: 2
```

Summing Array Entries: Version 1

```
public class ArraySum {
  public static void main(String[] args) {
    double[] numbers = { 1.1, 2.2, 3.3 };
    System.out.println("[v1] Sum of {1.1,2.2,3.3}=" +
                       arraySum1(numbers));
  }

  public static double arraySum1(double[] nums) {
    double sum = 0;
    for(double num: nums) {
      sum = sum + num; // Or sum += num
    }
    return(sum);
  }
```

```
[v1] Sum of {1.1,2.2,3.3}=6.6
```

Summing Array Entries: Version 2

```
public class ArraySum {
  public static void main(String[] args) {
    double[] numbers = { 1.1, 2.2, 3.3 };
    System.out.println("[v2] Sum of {1.1,2.2,3.3}=" +
                       arraySum2(numbers));
  }

  public static double arraySum2(double[] nums) {
    double sum = 0;
    for(int i=0; i<nums.length; i++) {
      sum = sum + nums[i];
    }
    return(sum);
  }
```

```
[v2] Sum of {1.1,2.2,3.3}=6.6
```

Summing Array Entries: Version 3

```
public class ArraySum {
  public static void main(String[] args) {
    double[] numbers = { 1.1, 2.2, 3.3 };
    System.out.println("[v3] Sum of {1.1,2.2,3.3}=" +
                       arraySum3(numbers));
  }

  public static double arraySum3(double[] nums) {
    double sum = 0;
    int i=0;
    while(i<nums.length) {
      sum = sum + nums[i];
      i++; // Or i = i + 1, or i += 1
    }
    return(sum);
  }
```

```
[v3] Sum of {1.1,2.2,3.3}=6.6
```

Summing Array Entries: Version 4

```
public class ArraySum {
  public static void main(String[] args) {
    double[] numbers = { 1.1, 2.2, 3.3 };
    System.out.println("[v4] Sum of {1.1,2.2,3.3}=" +
                       arraySum4(numbers));
  }

  public static double arraySum4(double[] nums) {
    double sum = 0;
    int i=0;
    do {
      sum = sum + nums[i];
      i++;
    } while(i<nums.length);
    return(sum);
  }
```

```
[v4] Sum of {1.1,2.2,3.3}=6.6
```

Defining Multiple Methods in Single Class

```
public class LoopTest {
  public static void main(String[] args) {
    String[] test = {"This", "is", "a", "test"};
    listEntries(test);
    listNums1(5);
    listNums2(6);
    listNums3(7);
  }

  public static void listEntries(String[] entries) {...}
  public static void listNums1(int max) {...}
  public static void listNums2(int max) {...}
  public static void listNums3(int max) {...}
}
```

These methods say "static" because they are called directly from "main". In the upcoming sections on OOP, we will explain what "static" means and why most regular methods do not use "static". But for now, just note that methods that are *directly* called by "main" must say "static".

Indentation: Blocks that are Nested More Should be Indented More

Yes

```
blah;
blah;
for(...) {
  blah;
  blah;
  for(...) {
    blah;
    blah;
  }
}
```

No

```
blah;
blah;
for(...) {
blah;
blah;
for(...) {
blah;
blah;
}
}
```

Indentation: Blocks that are Nested the Same Should be Indented the Same

Yes

```
blah;
blah;
for(...) {
  blah;
  blah;
  for(...) {
    blah;
    blah;
  }
}
```

No

```
blah;
 blah;
for(...) {
   blah;
  blah;
  for(...) {
    blah;
     blah;
  }
  }
```

Indentation: Number of Spaces and Placement of Braces is a Matter of Taste

OK

```
blah;
blah;
for(...) {
   blah;
   blah;
   for(...) {
      blah;
      blah;
   }
}
```

OK

```
blah;
blah;
for(...) {
      blah;
      blah;
      for(...) {
            blah;
            blah;
      }
}
```

OK

```
blah;
blah;
for(...)
{
   blah;
   blah;
   for(...)
   {
      blah;
      blah;
   }
}
```

Some organizations or projects make coding-style documents that all developers in the organization or on the project should follow. For example, the one for Google can be found at https://google.github.io/styleguide/javaguide.html. Although I personally follow almost all of those stylistic conventions, I am skeptical about how necessary or even valuable it is to enforce this on everyone in an organization or project.

If Statements: One or Two Options

- **Single option**

```
if (booleanExpression) {
   statement1;
   ...
   statementN;
}
```

- **Two options**

```
if (booleanExpression) {
   ...
} else {
   ...
}
```

The value inside parens must be strictly boolean (i.e., true or false), unlike C, C++, and JavaScript.

A widely accepted best practice is to use the braces even if there is only a single statement inside the if or else.

If Statements: More than Two Options

- **Multiple options**

```
if (booleanExpression1) {
   ...
} else if (booleanExpression2) {
   ...
} else if (booleanExpression3) {
   ...
} else {
   ...
}
```

Switch Statements

- **Example**

```
int month = ...;
String monthString;
switch(month) {
  case 0: monthString = "January";  break;
  case 1: monthString = "February"; break;
  case 2: monthString = "March";    break;
  ...
  default: monthString = "Invalid month"; break;
}
```

- **Syntax is mostly like C and C++**
  - Types can be primitives, enums, and (Java 7 and later) Strings

Boolean Operators

- **==, !=**
  - Equality, inequality. In addition to comparing primitive types, == tests if two objects are identical (the same object), not just if they appear equal (have the same fields). More details when we introduce objects.
- **<, <=, >, >=**
  - Numeric less than, less than or equal to, greater than, greater than or equal to.
- **&&, ||**
  - Logical AND, OR. Both use short-circuit evaluation to more efficiently compute the results of complicated expressions

```
if ((n > 5) && (n < 8)) { doFor6or7(...); }
```

- **!**
  - Logical negation. For example, `if (!(x < 5))` is the same as `if (x >= 5)`

Example: If Statements

```
public static int max(int n1, int n2) {
   if (n1 >= n2) {
      return(n1);
   } else {
      return(n2);
   }
}
```

Strings: Basics

- **Overview**
  - String is a real class in Java, not an array of characters as in C++
  - The String class has a shortcut method to create a new object: just use double quotes

```
String s = "Hello";
```

    - Differs from normal classes, where you use **`new`** to build an object
- **Use `equals` to compare strings**
  - Never use == to test if two Strings have same characters!

Using == to Compare Strings (Wrong!)

```
public class CheckName1 {
  public static void main(String[] args) {
    if (args.length == 0) {
      System.out.println("Nobody");
    } else if (args[0] == "Marty") {
      System.out.println("Hi, Marty");
    } else {
      System.out.println("Hi, stranger");
    }
  }
}
```

This always prints “Hi, stranger”, even if the first command line argument is “Marty”.

Using equals to Compare Strings (Right!)

```
public class CheckName2 {
  public static void main(String[] args) {
    if (args.length == 0) {
      System.out.println("Nobody");
    } else if (args[0].equals("Marty")) {
      System.out.println("Hi, Marty");
    } else {
      System.out.println("Hi, stranger");
    }
  }
}
```

Strings: Methods

- **Methods to call *on* a String**
  - contains, startsWith, endsWith, indexOf, substring, split, replace, replaceAll, toUpperCase, toLowerCase, equalsIgnoreCase, trim, isEmpty, etc.
    - For replacing, can use regular expressions, not just static strings
  - Example

```
String word = "...";
if (word.contains("q")) { ... }
```

- **Static methods in String class**
  - String.format, String.join, String.valueOf, etc.
  - Example

```
String numberAsString = String.valueOf(17);
```

Building Arrays: One-Step Process

- **Declare and allocate array in one fell swoop**

```
type[] var = { val1, val2, ... , valN };
```

- **Examples:**

```
int[] values = { 10, 100, 1_000 };
String[] names = {"Joe", "Jane", "Juan"};
Point[] points = { new Point(0, 0),
                   new Point(1, 2),
                   new Point(3, 4) };
```

Minor note: in Java 7 and later, underscores are ignored in numbers, so 1_000 above is the same as 1000.

Building Arrays: Two-Step Process

- **Step 1: allocate an empty array (really array of references):**

  Type[] var = new *Type*[size];

  – E.g.:

```
int[] primes = new int[x];        // x is positive integer
String[] names = new String[someInteger];
```

- **Step 2: populate the array**

```
primes[0] = 2;
primes[1] = 3;
primes[2] = 5;
primes[3] = 7;
```

etc. (or use a loop to fill in values)

```
names[0] = "Joe";
names[1] = "Jane";
names[2] = "Juan";
names[3] = "John";
```

etc. (or use a loop to fill in values)

Default Array Values

- **If you fail to populate an entry**
  - Default value is 0 for numeric arrays

```
int[] nums = new int[7];
int value = 3 + nums[2];   // value is 3
```

  - Default value is **`null`** for Object arrays

```
String[] words = new String[7];
System.out.println(words[2]); // Prints null
if (words[3].contains("q") { ... }
      // Crashes with NullPointerException
```

Two-Step Process: Example 1

```
public static Circle[] makeCircles1(int numCircles) {
  Circle[] circles = new Circle[numCircles];
       // Empty array of proper size
  for(int i=0; i<circles.length; i++) {
    circles[i] = new Circle(Math.random() * 10);
       // Populate array
  }
  return(circles);
}
```

This approach is correct!

Two-Step Process: Example 2

```
public static Circle[] makeCircles2(int numCircles) {
  Circle[] circles = new Circle[numCircles];
       // Empty array of proper size
  for(int i=0; i<circles.length; i++) {
    circles[i].setRadius(Math.random() * 10);
         // NullPointerException
  }
  return(circles);
}
```

This approach fails: the call to setRadius crashes with NullPointerException because circles[i] is null.

Two-Step Process: Example 3

```
public static Circle[] makeCircles3(int numCircles) {
  Circle[] circles = new Circle[numCircles];
  for(Circle c: circles) {
    c = new Circle(Math.random() * 10);
        // Fails to store c in array
  }
  return(circles);
      // Array still contains only null pointers
}
```

This approach fails: array is still empty after the loop.

Multidimensional Arrays

- **Multidimensional arrays**
  - Implemented as arrays of arrays

```
int[][] twoD = new int[64][32];

String[][] cats = {{ "Caesar",  "blue-point" },
                  { "Heather", "seal-point" },
                  { "Ted",     "red-point"  }};
```

- **Note:**
  - Number of elements in each row need not be equal

```
int[][] irregular = { { 1 },
                      { 2, 3, 4},
                      { 5 },
                      { 6, 7 } };
```

TriangleArray: Example

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

    int[][] triangle = new int[10][];

    for(int i=0; i<triangle.length; i++) {
      triangle[i] = new int[i+1];
    }

    for (int i=0; i<triangle.length; i++) {
      for(int j=0; j<triangle[i].length; j++) {
        System.out.print(triangle[i][j]);
      }
      System.out.println();
    }
  }
}
```

TriangleArray: Result

```
> java TriangleArray

0
00
000
0000
00000
000000
0000000
00000000
000000000
0000000000
```

Basic Mathematical Operators

- **+, -, *, /, %**
  - Addition, subtraction, multiplication, division, mod
    - mod means remainder, so 3 % 2 is 1.
- **num++, ++num**
  - Means add one to (after/before returning value)

```
int num = 3;
num++;
   // num is now 4
```

  - Usage
    - For brevity and tradition, but no performance benefit over simple addition
- **Warning**
  - Be careful with / on **`int`** and **`long`** variables (rounds off)

Basic Mathematical Methods: Usage

- **Static methods in the Math class**
  - So you call Math.cos(...), Math.random(), etc.
    - Most operate on double-precision floating point numbers
  - Examples

```
double eight = Math.pow(2, 3);
double almostZero = Math.sin(Math.PI);
double randomNum = Math.random();
```

  - In the JUnit section, we will cover static imports that let you skip the class name
  - Most developers do not use static imports with the Math class, but a few do. Quick example:

```
import static java.lang.Math.*;
...
double d1 = cos(...);   // Instead of Math.cos(...)
double d2 = sin(...);   // Instead of Math.sin(...)
double d3 = random();   // Instead of Math.random(...)
```

Basic Mathematical Methods

- **Simple operations: Math.pow(), etc.**
  - Math.pow (x^y), Math.sqrt ($\sqrt{x}$), Math.cbrt, Math.exp (e^x), Math.log ($\log_e$), Math.log10

```
double twoToThird = Math.pow(2, 3); // Returns 8.0
double squareRootOfTwo = Math.sqrt(2.0); // 1.414...
```

- **Trig functions: Math.sin(), etc.**
  - Math.sin, Math.cos, Math.tan, Math.asin, Math.acos, Math.atan
    - Args are in radians, not degrees, (see Math.toDegrees and Math.toRadians)
- **Rounding and comparison: Math.round(), etc.**
  - Math.round/Math.rint, Math.floor, Math.ceiling, Math.abs, Math.min, Math.max
- **Random numbers: Math.random()**
  - Math.random() returns double from 0 inclusive to 1 exclusive

```
double ranFrom0to1 = Math.random();
double ranFrom2to12 = 2.0 + (Math.random() * 10);
```

  - See Random class for more control over randomization

Common Use of Math.random

- **To randomly invoke different operations**
  - Especially useful for testing

```
for(int i=0; i<10; i++) {
  if (Math.random() < 0.5) {
    doFirstOperation();  // 50% chance
  } else {
    doSecondOperation(); // 50% chance
  }
}
```

More Mathematical Options

- **Special constants**
  - Double.POSITIVE_INFINITY
  - Double.NEGATIVE_INFINITY
  - Double.NAN
  - Double.MAX_VALUE
  - Double.MIN_VALUE
- **Unlimited precision libraries**
  - BigInteger, BigDecimal
    - Contain basic math operations like add, pow, mod, etc.
    - BigInteger also has isPrime

In Real Applications, use GUI

- **Practice: use approaches shown next**
  - Command line args
  - JOptionPane
  - Scanner
- **Real life: desktop and phone apps**
  - Collect input within Java app using textfields, sliders, dropdown menus, etc.
  - Convert to numeric types with Integer.parseInt, Double.parseDouble, etc.
- **Real life: Web apps (JSF2 and PrimeFaces)**
  - Collect input on browser with textfields, sliders, popup calendars, etc.
  - Java will convert automatically for simple types
  - You can set up converters for complex types
    - Details on JSF and PrimeFaces: http://www.coreservlets.com/JSF-Tutorial/jsf2/

Reading Strings from Users

- **Option 1: use command-line argument**

```
String input = args[0];
```

  - First verify that args.length > 0

- **Option 2: use JOptionPane**

```
String input = JOptionPane.showInputDialog("Number:");
```

- **Option 3: use Scanner**

```
Scanner inputScanner = new Scanner(System.in);
String input = inputScanner.next();
```

Converting Strings to Numbers

- **To int: Integer.parseInt**

```
String input = ...;
int num = Integer.parseInt(input);
```

- **To double: Double.parseDouble**

```
String input = ...;
double num = Double.parseDouble(input);
```

- **With Scanner**
  - Use scanner.nextInt(), scanner.nextDouble()
- **Warning**
  - In real life, you must handle the case where the input is not a legal number.
    - Idea shown without explanation in Input1Alt class
    - Try/catch blocks and exception handling covered in the section on simple graphics

Command-Line Args

```
public class Input1 {
  public static void main(String[] args) {
    if (args.length > 1) {
      int num = Integer.parseInt(args[0]);
      System.out.println("Your number is " + num);
    } else {
      System.out.println("No command-line args");
    }
  }
}
```

Open command window and navigate to folder containing class

```
> java Input1 7
Your number is 7
```

Preview of Error Checking (Explained in Section on Simple Graphics)

```
public class Input1Alt {
  public static void main(String[] args) {
    if (args.length > 1) {
      try {
        int num = Integer.parseInt(args[0]);
        System.out.println("Your number is " + num);
      } catch(NumberFormatException e) {
        System.out.println("Input is not a number");
      }
    } else {
      System.out.println("No command-line arguments");
    }
  }
}
```

Open command window and navigate to folder containing class

```
> java Input1Alt seven
Input is not a number
> java Input1Alt 7
Your number is 7
```

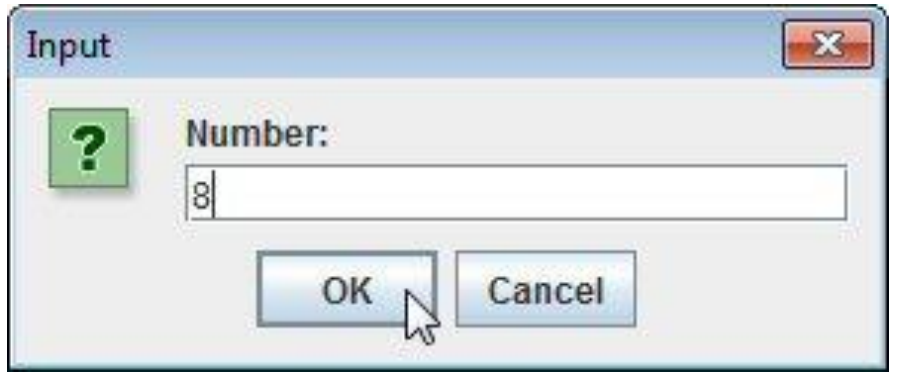
JOptionPane

```
... (package statement)
import javax.swing.*;

public class Input2 {
  public static void main(String[] args) {
    String input = JOptionPane.showInputDialog("Number:");
    int num = Integer.parseInt(input);
    System.out.println("Your number is " + num);
  }
}
```

Run from Eclipse (R-click, Run As → Java Application),
enter 8 in popup window
Result in Eclipse Console:

```
Your number is 8
```



Scanner

```
... (package statement)
import java.util.*;

public class Input3 {
  public static void main(String[] args) {
    System.out.print("Number: ");
    Scanner inputScanner = new Scanner(System.in);
    int num = inputScanner.nextInt();
    System.out.println("Your number is " + num);
  }
}
```

Run from Eclipse (R-click, Run As → Java Application),
enter 9 after "Number:" prompt in Eclipse Console. Next line:

```
Your number is 9
```

Summary

- **Basics**
  - Loops, conditional statements, and array access is similar to C/C++
    - But additional “for each” loop: for(String s: someStrings) { … }
  - Indent your code for readability
  - **`String`** is a real class in Java
    - Use **`equals`**, not ==, to compare strings
- **Allocate arrays in one step or in two steps**
  - If two steps, loop down array and supply values
- **Use Math.*blah*() for simple math operations**
  - Math.random, Math.sin, Math.cos, Math.pow, etc.
- **Simple input from command window**
  - Use command line for strings supplied at program startup
  - Use JOptionPane or Scanner to read values after prompts
    - Neither is very important for most real-life applications