8352 – Java From the Very Beginning 1 of 3

Charlote Sutton
Choice Hotels International
charlote_sutton@choicehotels.com

Objectives

- What is Java?
- What Java can do?
- Explore the Eclipse development environment.
- Write simple Java programs.

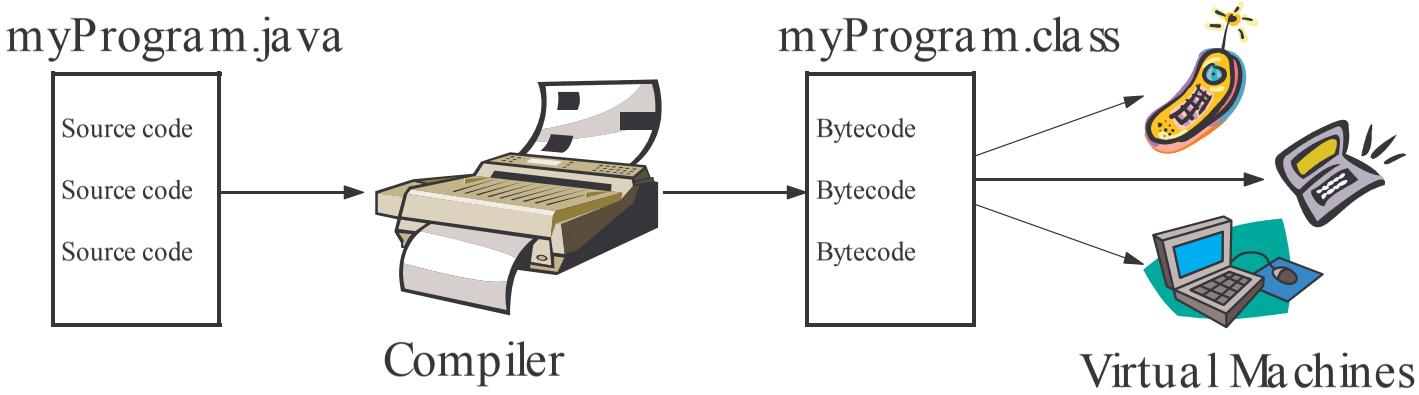
What is Java?

- A platform
  - Software only
  - Runs on top of hardware platforms
  - Two components:
    - JVM – Java Virtual Machine
    - API – Application Programming Interface
- A programming language
  - Compiled and Interpreted

Java Bytecode

- Instructions for JVM
- Write once, run anywhere
  - Compiled bytecode is platform independent.
  - If a platform--whether it's Windows, UNIX, MacOS, or an Internet browser--has the JVM, it can understand these bytecodes.
  - Any device capable of running Java will be able to interpret bytecode into something it can run.

Write > Compile > Interpret > Run



What can Java do?

- Types of Java applications
  - Applets
  - Applications
  - JavaBeans
  - Servers
  - Servlets

The Java API

- Included in Java platform
- Prewritten code
  - Organized into packages of similar topics

The Core API – The Essentials

- Objects
- Strings
- Threads
- Numbers
- Input and output
- Data structures
- System properties
- Date and time

More API Packages

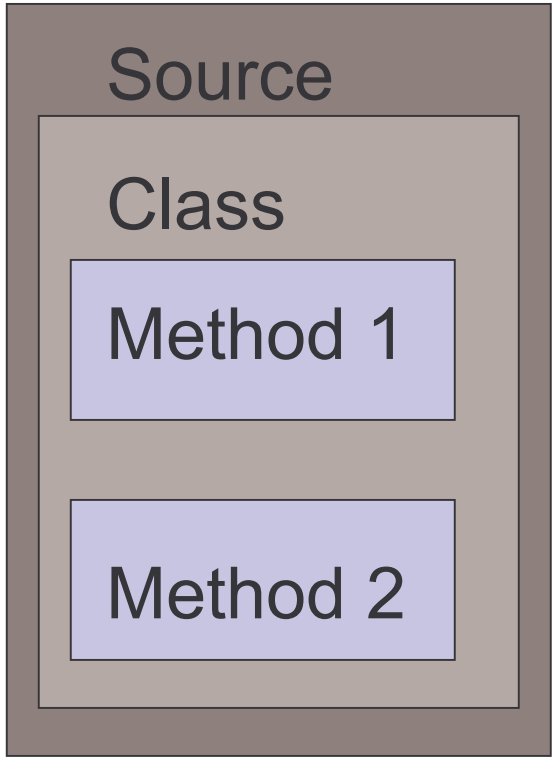
- Applets
- Internationalization
- Security
- Graphical User Interface
- Serialization
- Java Database Connectivity (JDBC)

Benefits of Java

- Get started quickly
- Write less code
- Write better code
- Write programs faster
- Avoid platform dependencies
- Write once, run anywhere
- Distribute software more easily

Code Structure in Java

- Source file
- Class
- Methods
- Statements



Anatomy of a class

```
public class MyFirstApp {
    public static void main (String[] args) {
        System.out.print("I rule!");
    }
}
```

Anatomy of a class

Public so everyone
can access it.

```
public class MyFirstApp {
    public static void main (String[] args) {
        System.out.print("I rule!");
    }
}
```

Anatomy of a class

Defines this
code as a class.

```
public class MyFirstApp {
    public static void main (String[] args) {
        System.out.print("I rule!");
    }
}
```

Anatomy of a class

The name of this class.

```
public class MyFirstApp {
    public static void main (String[] args) {
        System.out.print("I rule!");
    }
}
```

Anatomy of a class

Opening curly brace
marks start of class.

```
public class MyFirstApp {
    public static void main (String[] args) {
        System.out.print("I rule!");
    }
}
```

Anatomy of a class

```
public class MyFirstApp {
    public static void main (String[] args) {
        System.out.print("I rule!");
    }
}
```

Method code

Anatomy of a class

```
public class MyFirstApp {
    public static void main (String[] args) {
        System.out.print("I rule!");
    }
}
```

Closing curly brace marks end of class.

Anatomy of a method

```
public static void main (String[] args) {
        System.out.print("I rule!");
    }
```

Anatomy of a method

Public so anyone can access this method.

```
public static void main (String[] args) {
        System.out.print("I rule!");
    }
```

Anatomy of a method

This is a static method. You will learn more about this later.

```
public static void main (String[] args) {
        System.out.print("I rule!");
    }
```

Anatomy of a method

This is the return type.

Void = returns nothing.

```
public static void main (String[] args) {
        System.out.print("I rule!");
    }
```

Anatomy of a method

The name of the method.

```
public static void main (String[] args) {
        System.out.print("I rule!");
    }
```

Anatomy of a method

The arguments to the method.

```
public static void main (String[] args) {
        System.out.print("I rule!");
    }
```

Anatomy of a method

```
public static void main (String[] args) {
        System.out.print("I rule!");
    }
```

Opening curly brace marks start of method.

Anatomy of a method

```
public static void main (String[] args) {
        System.out.print("I rule!");
    }
```

The one statement in this method. Every statement must end with a semicolon.

Anatomy of a method

```
public static void main (String[] args) {
        System.out.print("I rule!");
    }
```

Closing curly brace
marks end of method.

Basic Java Syntax

- Comments
- Variables and Data Types
- Primitive Data Types
- Reference Data Types
- Scope
- Operators
- Expressions
- Arrays
- Strings

Comments

- /* text */
- /** documentation */
- // text

Variables and data types

- Variable declaration
  - Name
    - Can begin with letter, dollar sign, or underscore
    - Followed by letters, underscores, dollar signs, or digits
    - Convention isUpper
  - Type
    - Java's compiler cares about type
    - Determines value and operations
- Two kinds of variables
  - Primitive
  - Object Reference

Primitive types

- Hold fundamental values (simple bit patterns)
  - Integers
  - Booleans
  - Floating point numbers
  - Characters

Primitive types

Type	Bit Depth	Value Range
boolean	varies	true or false
char	16 bits	0 to 65535
byte	8 bits	-128 to 127
short	16 bits	-32768 to 32768
int	32 bits	-2147483648 to 2147483647
long	64 bits	-huge to huge
float	32 bits	varies
double	64 bites	varies

Reference types

- Anything that is not primitive
  - Objects
    - Strings
    - Arrays
    - Classes
    - Interfaces

Scope

- Definition
  - The region of a program within which the variable can be referred to by its simple name.
  - Scope also determines when the system creates and destroys memory for the variable.

Scope

- The location of the variable declaration within your program establishes its scope and places it into one of these four categories:
  - Member variable
  - Local variable
  - Method parameter
  - Exception-handler parameter

Scope

Member variable scope

```
class myClass {
    char someCharacter;

    public void myMethod (parameters) {
        some code here;

        int someNumber;

        catch (parameters){
            some code here;
        }
    }
}
```

Scope

```
class myClass {
    char someCharacter;

    public void myMethod (parameters) {
        some code here;

        int someNumber;

        catch (parameters){
            some code here;
        }
    }
}
```

Method Parameter scope

Scope

```
class myClass {
    char someCharacter;

    public void myMethod (parameters) {
        some code here;

        int someNumber;

        catch (parameters){
            some code here;
        }
    }
}
```

Local variable scope

Scope

```
class myClass {
    char someCharacter;

    public void myMethod (parameters) {
        some code here;

        int someNumber;

        catch (parameters){
            some code here;
        }
    }
}
```

Exception-Handler variable scope

Operators - Arithmetic

Operator	Use	Description
+	op1 + op2	Adds op1 and op2
-	op1 - op2	Subtracts op2 from op1
*	op1 * op2	Multiplies op1 by op2
/	op1 / op2	Divides op1 by op2
%	op1 % op2	Computes remainder of dividing op1 by op2

Operators – Increment/Decrement

Operator	Use	Description
++	op++	Increments op by 1; evaluates to the value of op before it was incremented
++	++op	Increments op by 1; evaluates to the value of op after it was incremented
--	op--	Decrements op by 1; evaluates to the value of op before it was decremented
--	--op	Decrements op by 1; evaluates to the value of op after it was decremented

Operators – Relational

Operator	Use	Returns true if
>	op1 > op2	op1 is greater than op2
>=	op1 >= op2	op1 is greater than or equal to op2
<	op1 < op2	op1 is less than op2
<=	op1 <= op2	op1 is less than or equal to op2
==	op1 == op2	op1 and op2 are equal
!=	op1 != op2	op1 and op2 are not equal

Operators – Conditional

Operator	Use	Returns true if
&&	op1 && op2	op1 and op2 are both true, conditionally evaluates op2
||	op1 || op2	either op1 or op2 is true, conditionally evaluates op2
!	! Op	op is false
&	op1 & op2	op1 and op2 are both true, always evaluates op1 and op2
|	op1 | op2	either op1 or op2 is true, always evaluates op1 and op2
^	op1 ^ op2	if op1 and op2 are different--that is if one or the other of the operands is true but not both

Operators - Assignment

Operator	Use	Equivalent to
=	op1 = op2	assign op1 to the value in op2
+=	op1 += op2	op1 = op1 + op2
-=	op1 -= op2	op1 = op1 - op2
*=	op1 *= op2	op1 = op1 * op2
/=	op1 /= op2	op1 = op1 / op2
%=	op1 %= op2	op1 = op1 % op2
&=	op1 &= op2	op1 = op1 & op2
|=	op1 |= op2	op1 = op1 | op2

Expressions

- Series of variables, operations and method calls that evaluate to a single expression
  - Use parenthesis to specify precedence

```
int someNum = 6;
int anotherNum;
anotherNum = someNum -1;
anotherNum = (sumNum - 3) * 2;
anotherNum = someNum - 3 * 2;
```

Strings

- The String class is included in the java.lang.Object package that is available automatically with a Java download.
- The String class represents character strings.
- When you declare and use a String, you are actually using an instance of the String class.
- Basic use of a String

```
String s = "Hello World! ";
String t = "Look at me.";
System.out.println(s + t);
```

Hello World! Look at me.

Arrays

- The Array class is included in the java.lang.Object package.
- The Array class contains various methods for manipulating arrays (such as sorting and searching).
- Here is a simple example:

```
int[] nums;
nums = new int[3];

nums[0] = 1;
nums[1] = 2;
nums[2] = 3;
```

Anatomy of an Array

```
int[] nums;
```

Declares an array of int's named "nums."

```
nums = new int[3];

nums[0] = 1;
nums[1] = 2;
nums[2] = 3;
```

Anatomy of an Array

```
int[] nums;

nums = new int[3];

nums[0] = 1;
nums[1] = 2;
nums[2] = 3;
```

Instantiates an Array object with the key word "new".

Anatomy of an Array

```
int[] nums;

nums = new int[3];

nums[0] = 1;
nums[1] = 2;
nums[2] = 3;
```

Gives the Array object a length of three.

Anatomy of an Array

```
int[] nums;

nums = new int[3];

nums[0] = 1;
nums[1] = 2;
nums[2] = 3;
```

Assigns the variable, "nums," to be the object reference variable – or remote control.

Anatomy of an Array

```
int[] nums;

nums = new int[3];

nums[0] = 1;
nums[1] = 2;
nums[2] = 3;
```

Gives each element a value.

Eclipse Overview – Demonstration

- Create New Project
- Java perspective
- Edit window
- Outline window
- Problems window

Create New Class

- Main method checkbox
  - Class shell created automatically
- Hello World!
- Code auto-complete
- Syntax checking while typing
- Save
- Run Hello World
- Run as Java Application

Configurations

- Select Java Application -> New Name
- Main tab
  1. Project -> Browse
  2. Main class -> Search
  3. Choose Main Type
  4. Run

Debug Hello World

- Debug tab
- Source tab
- Variables tab
- Console tab
- Tasks tab

Lab exercises

- Marathon
- Phrase-O-Matic