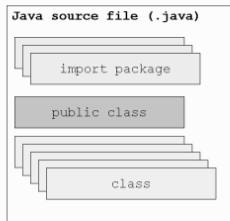


Introduction to Java Programming

Basic Syntax

Understand

Basic source file structure



A Java source file contains a series of import requests, a single distinguished public class, and an optional set of additional classes. Source file name is the same as the name of the public class.

Hello World

```
/**  
 * Simple "Hello World" text based application  
 */
```

```
import java.io.*;
```

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

Keywords & Identifiers

- Keywords are words that have specific meaning in Java (ex. `public`, `import`, etc.)
- An identifier is a name given to a variable, class or method
 - Can contain a-z, A-Z, 0-9, `_`, `$`, but can not start with 0-9
 - Must not be a keyword
 - Case sensitive
 - Of any length

Keywords * denotes keywords that are reserved but not used

- | | | | |
|-------------|--------------|----------|-----------|
| ■ abstract | extends | native | this |
| ■ boolean | false | new | throw |
| ■ break | final | null | throws |
| ■ byte | finally | package | transient |
| ■ case | float | private | true |
| ■ catch for | protected | try | |
| ■ char | goto* | public | void |
| ■ class | if | return | volatile |
| ■ const* | implements | short | while |
| ■ continue | import | static | |
| ■ default | instanceof | strictfp | |
| ■ do | int | super | |
| ■ double | interface | switch | else |
| ■ long | synchronized | | |

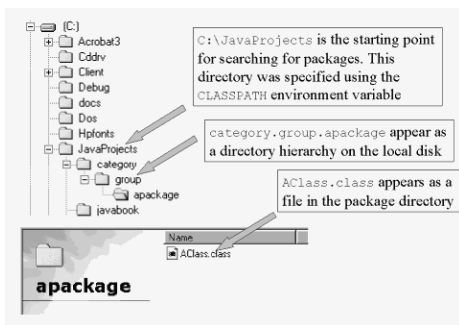
import

- Some pre-defined classes exist for your use in creating Java programs.
- Pre-defined classes are grouped together into “packages” for distribution, ie. `java.io`
- The import statement may have many names separated by period
 - `import java.awt.Graphics;`
 - `import category.group.apackage.AClass;`
 - `import java.io.*;`
- The asterisk (*) indicates that you wish to have access to all of the pre-defined classes in the `java.io` package
- Import statements with the (*) are not recursive. Meaning, if we import `java.util.*`, and we want to use `java.util.jar.*`, the following will **not** include `java.util.jar.*`, even though it is located in `java.util`.
 - `import java.util.*;`

import

- The CLASSPATH environment variable (or the `-classpath` option on the compiler) states where the directories containing packages on the disk are located.
 - If the classpath is not set correctly, the JVM will not be able to locate your java class files, thus your program will not run.
- The package names correspond to a directory path to the file containing the classes desired, or a path to the classes in `.jar` or `.zip` files

import category.group.apackage.AClass;



Inside the “public class” : room to solve “the problem”

```
/**  
 * Simple “Hello World” text based application  
 */
```

```
import java.io.*;
```

Class Identifier – an arbitrary name

```
public class HelloWorldApp {  
    public static void main (String[] arg) {  
        System.out.println(“Hello World”);  
        // Now we attempt to tell our age in celsius
```

```
    }  
}
```

End Main

End Class

Main method – all apps **must** have a main method

"The Problem"

Problems and data are not presented in a manner directly understandable by the machine.

- Provide paychecks for all company employees.
- Simulate the effects of an earthquake on the proposed high-rise office complex.
- Assess effective power production of a variety of wind turbine generators.
- Track a small business' sales and inventory.
- Process income tax returns from state residents.
- Calculate age in Celsius**
- Add 13 numbers together.

Simple algorithm

- Formula: $celsius = (fahrenheit - 32) * 5 / 9$
- Java code:

```
float fahrenheitAge = 45;
double celsiusAge;
celsiusAge = (fahrenheitAge-32) * 5.0 / 9;
System.out.print("I am ");
System.out.print(celsiusAge);
System.out.println(" years old! (In
Celsius)");
```

What's an Algorithm?

- An algorithm are the logical steps taken to solve a problem.
 - Problem Domain:
 - Need a Cake
 - Algorithm:
 - 3 eggs
 - Tablespoon of oil
 - Stick of butter
 - Cup of Milk
 - 1) In a large mixing bowl,
 - 2)Preheat oven ...

Some Java rules

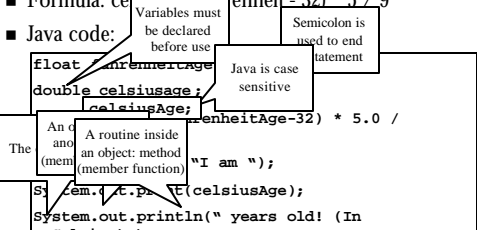
- Formula: $celsius = (fahrenheit - 32) * 5 / 9$
- Java code:

```

float fahrenheitAge;
double celsiusAge;
celsiusAge = (fahrenheitAge - 32) * 5.0 / 9;

// A routine inside an object: method (member function)
System.out.println("I am ");

// Semantics
System.out.println(" years old! (In Celsius)");
    
```

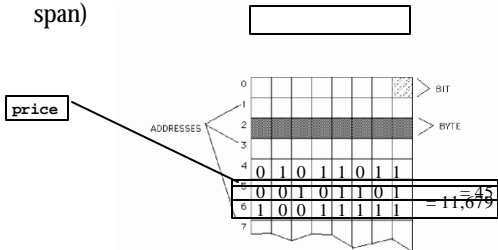


Data Types

- Reference data type (Object)
 - Classes (will learn later)
- Primitive data type
 - boolean (1 bit) true or false not number!
 - byte (8 bits) -128 .. +127 - INTEGER PRIMITIVE
 - char (16 bits) Unicode! Not 8 bits like C - INTEGER PRIMITIVE
 - short (16 bits) -32,768 .. +32,767 - INTEGER PRIMITIVE
 - int (32 bits) -2,147,483,648 .. +2,147,483,647
 - long (64 bits) -9223372036854775808 .. +9223372036854775807
 - float (32 bits) -3.4E38 .. +3.4E38 (6-7 significant figures accuracy)
 - double (64 bits) -1.7E308 .. +1.7E308 (13-14 digits accuracy)

Data Types

- Variables have name, type, size, value (and life span)



Date

```
celciusAge = (farenheitAge - 32) * 5/9;  
celciusAge = (farenheitAge - 32) * 5.0/9;  
celciusAge = (farenheitAge - 32) * (float)9;
```

- Variables have name, type, size, value (and life span)
 - int / int = ?
 - value + value = incorrect value ?
- Type casting
 - Syntax: (*Type*) original type
 - quotient = (double) sum / count;
 - intVar = (int) longValue;
 - longVar = (long) doubleValue;

Code sample of type conversion

```
int a;  
long b;  
double c;  
b = a; //this is ok b is larger  
a = b; //bad, a is smaller  
a = (int) b; //ok cast, may lose data  
c = b; //ok  
b = c; /*java does not support implicit  
conversion like this */  
b = (long) c; //but this is ok, but may lose  
data
```

Bug Alert: Type conversion side effects

```
byte a, b, c;  
a = b + c; /*this would produce a compile  
time error since b and c would be  
converted to ints and result of b + c  
would be an int */  
a = (byte)(b + c); //this is ok  
a += b; //this is ok
```

Widening Conversions

- When evaluating an arithmetic expression with two operands, the compiler converts primitives by widening according to these rules:
 - 1) If either is of type **double**, the other is converted to **double**
 - 2) Otherwise, if either is a **float**, the other is converted to **float**.
 - 3) Otherwise, if either is of type **long**, the other is converted to **long**.
 - 4) Otherwise, both operands are converted to **int**.

Arithmetic

Binary Operators – need two operands

- Assignment: =
 - Addition: a + b
 - Subtraction: c - 10
 - Multiplication: c * 0
 - Division: 45 / 5
 - No exponent operator
 - Modulus: 8 % 3
- * Remember the cast operator, (type), has precedence above *, /, %

Numeric Literals

- Dec 25 = Oct 31
`int myVar = 25; // decimal`
`int myVar = 031; // octal`
`int myVar = 0x19; // hex`
- Forcing integer to **long** storage: 1l, 1L, 79l
- Floating point literals: 563.84 == 5.6384e2
- Forcing to **float**: 3.142657e2f

Increment & Decrement operators -

- ++ (**a++** is same as **a=a+1**)
 - postincrement: **a++**;
 - preincrement: **++a**;
- - (**a--** is same as **a=a-1**)
 - also has postincrement and preincrement
- Example: **a=12; b=34; c=--a+b++**;
 - *PLEASE: Don't write code like this!*
 - **c = (--a) + b++**;
 - Why not **c = (a-1) + b; b = b+1**;
 - Why not **c = (--a) + (b++)**;

More assignment operators

- += (**a+=5** is **a=a+5**)
- -= (**a-=4** is **a=a-4**)
- *= (**a*=3** is **a=a*3**)
- /= (**a/=n** is **a=a/n**)
- %= (**b%=y** is **b=b%y**)
- Example:
 - **a=10; b=34**;
 - **a *= (b - 4)**;

Arithmetic Precedence

- Priority 1: () - negation
- Priority 2: * / %
- Priority 3: + -

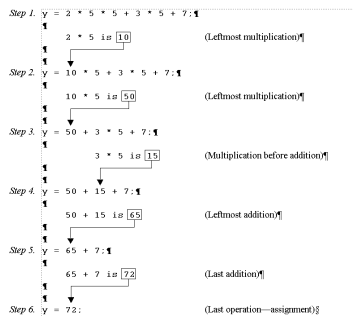
```
celsiusAge = fahrenheitAge - 32 * 5 / 9;  
celsiusAge = (fahrenheitAge - 32) * 5.0 / 9;
```

Quick Practice

- If you were a computer, what would you give as a result for:

$$y = 2 * 5 * 5 + 3 * 5 + 7;$$

Quick Practice Result



Quick Practice

- If you were a computer, what would you give as a result for:

$$y = 2 * 5 * 5 + 3 * 5 + 7;$$

- What is the calculation order of:

$$z = p * r \% q + w / x - y;$$

Java: $z = p * r \% q + w / x - y;$

6 1 2 4 3 5

Statement

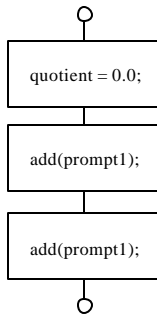
- Statement: the most basic building block, causes Java to perform a single action or operation (often ends w/ ;)
- Java code:

```
float fahrenheitAge = 45;
double celsiusAge;
celsiusAge = (fahrenheitAge * 5) / 9;
System.out.print("I am ");
System.out.print(celsiusAge);
System.out.println(" years old! (In Celsius)"); // Also statements
```

Special things on String

```
/** "Hello World" simple applet */
import java.applet.*;
import java.awt.Graphics;
public class HelloWorld extends Applet {
    public void paint (Graphics g){
        int number = 7;
        g.drawString("Hello World", 25, 25);
        g.drawString(number, 25, 50);
        g.drawString("" + number, 25, 75);
        g.drawString("sum of 2 & 9 = " + 2 + 9, 25, 99);
    }
}
```

Statement diagram



Break time

- After break
 - Coding Convention for Homework
 - Comments
 - JavaDoc

Hello World (applet)

```
/**
 * "Hello World" simple applet
 */
import java.applet.*;
import java.awt.Graphics;

public class HelloWorld extends Applet {
    public void paint (Graphics g){
        g.drawString("Hello World", 25, 25);
    }
}
```

HTML file for applet

```
<html>
<body>
  <applet code="HelloWorld.class" width=275
          height=55>

  </applet>
</body>
</html>
```

Conditional Statement: if()

- Syntax:
 - `if (data1 cond.operator data2) {`
 - `statement1;`
 - `statement2;`
 - ...
 - `}`

Conditional Operators

- Equality `a == b`
- Inequality `a != b`
- Greater than `a > b`
- Less than `a < b`
- Greater than or equal `a >= b`
- Lesser than or equal `a <= b`
 - Notice: works mainly with primitive data type

Event Driven Programming (Just enough to do the assignment)

- What is Event-Driven programming?
- Getting started with event-driven programming in Java:
 - Applet starts with `init()`, `start()` and `paint()` methods
 - `action()` trap users response to the object

Character Literals

- Single unicode character appear in `' '`
- Escape sequence
 - `\'` Single Quote `\"` Double Quote
 - `\\` Backslash `\b` Backspace
 - `\r` Carriage Ret. `\n` New Line
 - `\t` Horizontal Tab `\f` Form Feed
 - `\u`Unicode `\ddd` Octal char value

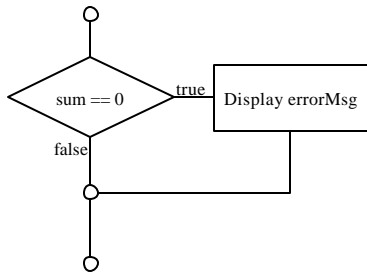
if structure (pick up from where we left off)

- Syntax:
 - `if (data1 cond.operator data2) {`
 - `statement1;`
 - `statement2;`
 - ...
 - `}`

Comments

- C style comment
 - `/* comment */`
 - `/* All character between /* and */ are ignored */`
- C++ style comment
 - `// comment after the // up to end of line`
- Java blend for javadoc documentation
 - `/** comment */`

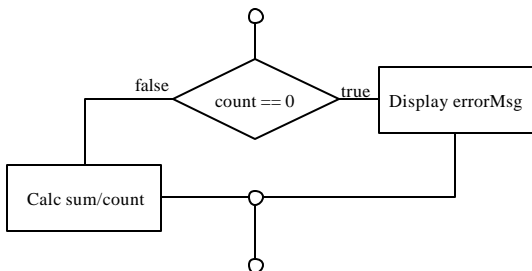
if structure diagram



if / else structure

- Syntax:
 - **if** (*logical condition*) {
 - *statement1*;
 - *statement2*;
 - ...
 - **} else** {
 - *statementA*;
 - *statementB*;
 - ...
 - }

if / else structure diagram



Common if problems

- `if (count == 0)`
 - `System.out.println("Divided-by-Zero");`
- `if (count != 0);`
 - `System.out.println("Good divisor");`
- `if (count != 0)`
 - `if (sum != 0)`
 - `System.out.println("Both numbers are not zero");`
- `else`
 - `System.out.println("Divided-by-Zero");`

Logical Operators

- `if ((count != 0) && (sum != 0))`
 - `System.out.println("Both numbers are not zero");`
- `else if (count==0)`
 - `System.out.println("Divided-by-Zero") error;`
- Logical AND `&&` truth table:
- Logical OR `||` truth table:
- Example: `question = (bb) || !(bb) ;`
- Short-circuit evaluation

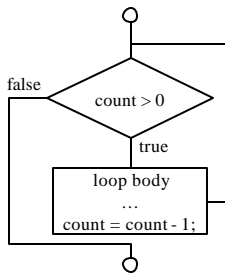
Bitwise Operators

- `&` (bitwise AND)
 - `00010101`
 - `&` `00000111`
- `|` (bitwise OR)
 - `00010101`
 - `&` `00000111`
- `^` (bitwise XOR)
 - `00010101`
 - `&` `00000111`

while structure

- Syntax:
 - **while** (*logical condition*) {
 - *statement1*;
 - *statement2*;
 - ... *loop body* ...
 - }

while structure diagram



while examples

- Counter control example:
 - **int** i = 10;
 - **while** (i > 0) {
 - System.out.println("counter = " + i);
 - i = i - 1;
 - }
- Sentinel control example:
 - **char** c = 'A';
 - **while** (c != 'Q') {
 - c = System.in.read();
 - System.in.skip(2);
 - }

After break: Java Application

- `public class AppName {`
 - `public static void main (String args[]) {`
 - ...
 - }
 - }
- Compile with **JavaC** as usual
- Run with **Java *AppName***

JDB: Java Debugger

- Compile with `javac -g ClassName.java`
- Run in debug mode
 - Use `jdb ClassName` to debug application.
 - Use `appletviewer -debug Container.html` to debug applet.
- Typical jdb commands: `help`, `exit`, `list`, `stop at <class id>:<line>`, `clear <class id>:<line>`, `step`, `cont`, `print`, `dump`
- jdb requires TCP/IP to work!

- brippe@fullcoll.edu – intro java
- <http://cis226.fullcoll.edu>
- OHHH!
- I forgot to mention, THIS CLASS IS SUPPOSED TO BE FUN!
