**Parts of a Java Program**

An overview

- At least one file
  - Realistic programs are composed of many files
  - The order of file contents are enforced by the compiler
    - package statement optional
    - import statement(s) optional
- At least one class
  - Realistic programs are composed of many classes
  - Only one public class per file
- At least one main method (for an application)
  - The code may contain multiple mains
  - Only one main will execute
  - public static void main(String[] args)

**Parts Of A Method**

Components of all methods

- Return type
- Method name
- Parameters (for all identifiers)
- Must be unique
- Parameters list
  - May be empty
  - Parameter type and name pair
- Body
  - Local variables
  - Statements
- "Local" variables are defined in a method and are not accessible or "visible" outside of the method

**Primitive (Intrinsic) Types**

Simple, built-in data types

<table>
<thead>
<tr>
<th>Type</th>
<th>Bytes</th>
<th>Range</th>
<th>Values</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>void</td>
<td>0</td>
<td>true, false</td>
<td>return type</td>
<td></td>
</tr>
<tr>
<td>byte</td>
<td>1</td>
<td>-128 to 127</td>
<td></td>
<td></td>
</tr>
<tr>
<td>char</td>
<td>2</td>
<td>0 to 255</td>
<td></td>
<td></td>
</tr>
<tr>
<td>short</td>
<td>2</td>
<td>-32768 to 32767</td>
<td></td>
<td></td>
</tr>
<tr>
<td>int</td>
<td>4</td>
<td>-2147483648 to 2147483647</td>
<td></td>
<td></td>
</tr>
<tr>
<td>long</td>
<td>8</td>
<td>-9.2233720368547758e+18 to 9.2233720368547758e+18</td>
<td></td>
<td></td>
</tr>
<tr>
<td>float</td>
<td>4</td>
<td>-3.4028236692093846e+38 to 3.4028236692093846e+38</td>
<td>6 sig digs</td>
<td></td>
</tr>
<tr>
<td>double</td>
<td>8</td>
<td>-1.7976931348623157e+308 to 1.7976931348623157e+308</td>
<td>15 sig digs</td>
<td></td>
</tr>
</tbody>
</table>

- Type determines size and interpretation of bits
- Size and byte-order are Java-defined
- Signed -2<sup>31</sup> to 2<sup>31</sup> - 1; Unsigned 0 to 2<sup>32</sup> - 1

**boolean Type**

Bivalued logic type

- Java keywords: boolean, true, false
- Cannot convert between booleans and ints
- Relational expressions return boolean values
  - != done = false; // assignment of a boolean value
  - != done = false; // assignment of a boolean value
  - != done = false; // true if e1 is false
  - != done = false; // true if e1 is not false
  - & & e1 & e2; // short circuit version
  - & & e1 & e2; // short circuit version
  - || e1 || e2; // short circuit version
  - || e1 || e2; // short circuit version
  - || e1 || e2; // short circuit version
  - || e1 || e2; // short circuit version
  - || e1 || e2; // short circuit version
  - || e1 || e2; // short circuit version
  - || e1 || e2; // short circuit version
  - || e1 || e2; // short circuit version
**char Type**

Character data based on the unicode encoding

- Two-byte data values
  - 65,536 characters ("u0000" to "uFFFF")
  - The first 256 are identical to the extended ASCII character set
- Escape sequences for common characters (see p. 15)
  - \b backspace \u0008
  - \t horizontal tab \u0009
  - \n newline (linefeed) \u000A
  - \r carriage return \u000D
  - " double quote \u0022
  - ' single quote \u0027
  - \ single backslash \u005C

**String Class**

java.lang.String

- Instantiation
  - String greeting = "Hello world"; // preferred
  - String greeting = new String("Hello world"); // not preferred
- Methods
  - greeting.compareTo("hello world"); // like strcmp
  - greeting.equals("hello world"); // boolean
  - greeting.equalsIgnoreCase("hello world"); // boolean
  - greeting.length(); // 11
  - greeting.substring(0,5); // "hel"
  - greeting.indexOf(0); // 0
  - greeting.indexOf("world"); // 6
  - greeting.concat("" from Java")
  - greeting.concat("" from Java"); // expression
  - greeting.concat("" from Java"); // statement

**Literal Values**

Definitions and assignments

```java
char c1 = 'A';
char c2 = '?'
int runningTotal = 0;
int byteMask = 0xff;
// hex assignment
int permissions = 0777;
// octal
float fontSize = 2.7f;
double pi = 3.14157;
double avogadro = 6.02e23;
boolean done = false;
String error = "Unable to open file for reading";
```

**Variables**

Name a region of memory

- Have a content and an address
  - A Java does not permit access to the address
  - The content can vary over time
  - Can contain just one value at a time

- Must be defined before use (text uses "declaration," p. 65)
  - Data type and name
  - Amount of memory to allocate
  - How to interpret the bits

```java
int i; // undefined 0x00000000 i is uninitialized
i = 10;
```

**Instance and Class Variables**

Variable scope

- Instance variables/fields
  - Belong to an instance (i.e., memory is allocated in an object)
  - Can be accessed throughout a class
- Class variables
  - Belong to the defining class (i.e., memory is independent of objects)
  - Created with the "static" keyword
- Local variables
  - Arguments passed to a method (chapter 3)
  - Variables defined in a method
- Symbolic, named, or manifest constants
  - final and often static
  - public static final double PI = 3.14159;

**Java Identifiers (Naming Things)**

AKA symbols

- Used to name
  - Packages
  - Classes
  - Methods
  - Variables

- Legal identifiers (names)
  - Begin with a letter, including unincide letters (A-Z, a-z, _, $)
  - Subsequent characters may be letters and digits
  - Cannot contain operators like +, -, *, /, %, etc.
  - Although $ is legal, it should be avoided
  - Cannot be a reserved keyword
  - Are case sensitive
  - Are unlimited in length
### Naming Convention / Style

Java uses camel notation.

<table>
<thead>
<tr>
<th>Java Element</th>
<th>Naming Strategy</th>
</tr>
</thead>
<tbody>
<tr>
<td>package</td>
<td>all lowercase letters and numbers</td>
</tr>
<tr>
<td>class</td>
<td>start with uppercase, rest mixed, no underscores</td>
</tr>
<tr>
<td>method</td>
<td>start with lowercase, rest mixed, no underscores</td>
</tr>
<tr>
<td>instance variables</td>
<td>start with lowercase, rest mixed, no underscores</td>
</tr>
<tr>
<td>named constants</td>
<td>uppercase letters and underscores</td>
</tr>
</tbody>
</table>

### Java Operators

#### Building Java™ Statements

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ ]</td>
<td>indexing</td>
<td>→</td>
</tr>
<tr>
<td>()</td>
<td>member selection</td>
<td>→</td>
</tr>
<tr>
<td>++, --</td>
<td>auto increment and decrement</td>
<td>→</td>
</tr>
<tr>
<td>!</td>
<td>logical negation</td>
<td>→</td>
</tr>
<tr>
<td>~</td>
<td>bitwise complement</td>
<td>→</td>
</tr>
<tr>
<td>+, -</td>
<td>unary</td>
<td>→</td>
</tr>
<tr>
<td>.</td>
<td>member selection</td>
<td>→</td>
</tr>
<tr>
<td>()</td>
<td>method invocation</td>
<td>→</td>
</tr>
<tr>
<td>new</td>
<td>object instantiation</td>
<td>→</td>
</tr>
<tr>
<td>instanceof</td>
<td>class membership (boolean)</td>
<td>→</td>
</tr>
<tr>
<td>* / %</td>
<td>multiplication, division, modulo</td>
<td>→</td>
</tr>
<tr>
<td>+, -</td>
<td>addition and subtraction</td>
<td>→</td>
</tr>
</tbody>
</table>

### Java Operators

#### Operator Examples

#### "Unusual" operators

- **Auto increment and decrement (see p. 75)**
  - Target must be a variable (i.e., an l-value)
  - `int i, j = 10;
  - `i = j++;` "i is 10, j is 11 "
  - `i++; `"i is 11, j is 11 "
  - `i = j--;` "i is 10, j is 9 "
  - `i = --j;` "j is 9, j is 9 "
  - `i++, ++i, i--;` - and --are legal (i.e., assignment is not required)
  - May be embedded in expressions
  - Often used in array indexes

- **Multiple assignment (i.e., the = operator returns a value)**
  - `i = j = k = 0;` "i equivalent to i = (j = (k = 0)) "

#### More unusual operators

- `?:`
  - `a ? b : c;` "if a is true, then b, else c "

- `op= (+=, -=, *=, /=, %=, <<=, >>=, >>>=)`
  - `variable = expression` "variable = variable op expression;"
  - `x += 10;`
  - `i = 2;`
  - `a /= b;`
  - `index %= size;`
  - `mask <<= 2;`

#### Automatic Type Promotions

Type conversions (casting) performed during calculations

- `char` → `short` → `int` → `long`
- `float` → `double`

- **Unmodified literals**
  - Ordinals are type int
  - 3 is an int
  - Floating point are type double
  - 3.0 is a double

- **Modified literals**
  - `F` indicates a float
  - 3.0F is a float
  - A leading "0" is an octal
  - An leading "0x" is a hexadecimal
  - 012 is octal for 10
  - 0xf is hex for 15
Type Casts

User-specified type conversions
- Required when converting from "wide" to "narrow" data types
  - Wider means a greater range, not number of bytes
    - `float` (4 bytes) is "wider" than `long` (8 bytes)
  - The wide type must encompass the full range of the narrow type
- `(type) expression`;
  - `int i = (int) 3.14;`
  - `int l = (long) 3.14;`
  - `double d = (double) 1.3;`
- Will the following compile?
  - `byte b1 = 5;`
  - `byte b2 = 10;`
  - `b1 = b1 + b2;`
  - `b1++;`

Libraries and Wrapper Classes

Classes in the `java.lang` package
- Java is a pure object-oriented language: All library features are defined in classes
  - `public class Math {
      public static final double PI = 3.14159; // class constant
      public final sin(double x) {...}
    }
  - Wrapper classes correspond to primitive types
    - `Integer` `Float` `Character` `Long` `Double` `Boolean`
    - Define methods for manipulating primitive data
    - Allows storing primitive data in collections

Strings ↔ Numbers

Strings and numbers are not "close enough" to cast
- Strings to Numbers
  - `int i = Integer.parseInt("123");`
  - `double x = Double.parseDouble("3.14159");`
- Numbers to Strings
  - `String s = Integer.toString(i);`
  - `String s = String.valueOf(n);`
  - `String str = String.format("The answer is %d", 7);`
- Concatenate with the empty string (no space between "")
  - `String s = "String.format\("The answer is %d", 7);"

Simple Console Output

Details and examples in `java.io.PrintWriter`
- Numerous overloaded methods
  - `println(char)` `print(char)`
  - `println(int)` `print(int)`
  - `println(double)` `print(double)`
  - `println(String)` `print(String)`
  - `println(Object)` `print(Object)`
- System.out.println("Please enter the maximum size: ");
- System.out.println("Current Count = " + count);
- System.err.println("ERROR: x = " + x + " y = " + y);
- System.out.print("Enter your choice: "); // no new-line

Formatted Printing with `printf`

Details and examples in `java.util.Formatter`
- Used to write information to the screen or to a file
  - `printf("format string", arg1, arg2, arg3 ...);`
- The format string is required; other arguments are optional
- Characters in the format string, except format specifiers, are printed exactly as they appear in the string
- Arguments can be constants, variables, expressions, objects, etc.
  - `%d` (int, short, long); `%f` (float, double); `%c` (char); `%s` (String); `%n` (the percent character); `%n` (platform-specific line separator)
- System.out.print("The answers are: %d and %dn", ans1, ans2);

for Loop

Definite loop: initialize-test-update (p. 79)
- Test at top (may not execute)
- Composed of three expressions
  - Any expression may be omitted
  - Expression 1 is the initializer
  - Executed only once
  - Expression 2 is the loop test
  - Loops while expression 2 is true
  - Tested after expr 1
  - Tested after expr 3
  - Expression 3 is the update
    - for (expr1; expr2; expr3) statement-1;
    - for (int i = 0; i < 10; i++) System.out.println();