Java Syntax
- Java Syntax for Instructions
  - Keywords
  - Operators
  - Punctuations
- Java Syntax for Expressing Data
  - Keywords
  - Symbolic Names
  - Data Types

Java Basics
- Java Application Structure
- Data Types, Variables, and Constants
- Expressions and Arithmetic Operators

Java Application Structure
- All programs consist of at least one class.
- See Example 2.1 SkeletonApplication for standard form of Java application
- Java source code file must have the same name as class with .java extension.

Identifiers - Symbolic Names
- Identifiers are used to name classes, variables, and methods
- Identifier Rules:
  - Must start with a "Java letter"
    - A - Z, a - z, _, $, and Unicode letters
  - Can contain essentially any number of Java letters and digits, but no spaces
  - Case sensitive!!
    - Number1 and number1 are different!
  - Cannot be keywords or reserved words
    - See Appendix A
Program Building Blocks

- The Statement
  - Performs some action
  - Terminates with a semicolon (;)
  - Can span multiple lines

Building Blocks - The Block

- The Block
  - 0, 1, or more statements
  - Begins and ends with curly braces {}
  - Can be used anywhere a statement is allowed.

Building Blocks - White Space

- Space, tab, newline are white space characters
- At least one white space character is required between a keyword and identifier
- Any amount of white space characters are permitted between identifiers, keywords, operators, and literals

Building Blocks - Comments

- Comments explain the program to yourself and others
- Block comments
  - Can span several lines
  - Begin with /*
  - End with */
  - Compiler ignores all text between /* and */
- Line comments
  - Start with //
  - Compiler ignores text from // to end of line

Data Types, Variables, and Constants

- We use Symbolic Names to refer to data
- We must assign a data type for very identifier (symbolic name)
- Declaring Variables
- Primitive Data Types
- Initial Values and Literals
- String Literals and Escape Sequences
- Constants

Data Types

- For all data, assign a name (identifier) and a data type
- Data type tells compiler:
  - How much memory to allocate
  - Format in which to store data
  - Types of operations you will perform on data
- Compiler monitors use of data
  - Java is a "strongly typed" language
- Java "primitive data types"
  - byte, short, int, long, float, double, char, boolean
Declaring Variables

- Every variable must be given a name and a data type
- Variables hold one value at a time, but that value can change
- Syntax:
  ```java
dataType identifier;
```
or
  ```java
dataType identifier1, identifier2, ...;
```
- Naming convention for variable names:
  - first letter is lowercase
  - embedded words begin with uppercase letter

Names of variables should be meaningful and reflect the data they will store
  - This makes the logic of the program clearer
- Don’t skimp on characters, but avoid extremely long names
- Avoid names similar to Java keywords

Java Primitive Data Types

- `byte, short, int, long, float, double, char, boolean`

![Diagram of primitive data types]

Integer Types - Whole Numbers

<table>
<thead>
<tr>
<th>Type</th>
<th>Size in Bytes</th>
<th>Minimum Value</th>
<th>Maximum Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>1</td>
<td>-128</td>
<td>127</td>
</tr>
<tr>
<td>short</td>
<td>2</td>
<td>-32,768</td>
<td>32,767</td>
</tr>
<tr>
<td>int</td>
<td>4</td>
<td>-2,147,483,648</td>
<td>2,147,483,647</td>
</tr>
</tbody>
</table>

Example declarations:
```java
int testGrade;
int numPlayers, highScore, diceRoll;
short xCoordinate, yCoordinate;
byte ageInYears;
long cityPopulation;
```}

Floating-Point Data Types

- Numbers with fractional parts

<table>
<thead>
<tr>
<th>Type</th>
<th>Size in Bytes</th>
<th>Minimum Value</th>
<th>Maximum Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>float</td>
<td>4</td>
<td>1.4E-45</td>
<td>3.4028235E38</td>
</tr>
<tr>
<td>double</td>
<td>8</td>
<td>4.9E-324</td>
<td>1.797693134862157E308</td>
</tr>
</tbody>
</table>

Example declarations:
```java
float salesTax;
double interestRate;
double paycheck, sumSalaries;
```}

char Data Type

- One Unicode character (16 bits - 2 bytes)

<table>
<thead>
<tr>
<th>Type</th>
<th>Size in Bytes</th>
<th>Minimum Value</th>
<th>Maximum Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>2</td>
<td>character</td>
<td>character</td>
</tr>
<tr>
<td></td>
<td></td>
<td>encoded as 0</td>
<td>encoded as FFFF</td>
</tr>
</tbody>
</table>

Example declarations:
```java
char finalGrade;
char newline, tab, doubleQuotes;
```
**boolean Data Type**

- Two values only:
  ```java
type
false
  ```
- Used for decision making or as "flag" variables
- Example declarations:
  ```java
boolean isEmpty;
boolean passed, failed;
  ```

---

**Assigning Values to Variables**

- Assignment operator `=`
  - Value on the right of the operator is assigned to the variable on the left
  - Value on the right can be a literal (text representing a specific value), another variable, or an expression (explained later)
- Syntax:
  ```java
dataType variableName = initialValue;
Or
dataType variable1 = initialValue1,
variable2 = initialValue2, …;
  ```

---

**Literals**

- `int`, `short`, `byte`
  Optional initial sign (`+` or `-`) followed by digits 0 – 9 in any combination.
  ```java
int testGrade = 100;
  ```
- `long`
  Optional initial sign (`+` or `-`) followed by digits 0–9 in any combination, terminated with an `L` or `l`.
  
  ***Use the capital `L` because the lowercase `l` can be confused with the number 1.***

---

**Floating-Point Literals**

- `float`
  Optional initial sign (`+` or `-`) followed by a floating-point number in fixed or scientific format, terminated by an `F` or `f`.
- `double`
  Optional initial sign (`+` or `-`) followed by a floating-point number in fixed or scientific format.

---

**char and boolean Literals**

- `char`
  - Any printable character enclosed in single quotes
  - A decimal value from 0 – 65535
  - "\m", where \m is an escape sequence. For example, \n represents a newline, and \t represents a tab character.
- `boolean`
  ```java
true or false
  ```
- See Example 2.2 Variables.java

---

**Assigning the Values of Other Variables**

- Syntax:
  ```java
dataType variable2 = variable1;
  ```
- Rules:
  1. `variable1` needs to be defined before this statement appears in the source code
  2. `variable1` and `variable2` need to be compatible data types; in other words, the precision of `variable1` must be lower than or equal to that of `variable2`. 
Compatible Data Types

Any type in right column can be assigned to type in left column:

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Compatible Data Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>byte</td>
</tr>
<tr>
<td>short</td>
<td>byte, short</td>
</tr>
<tr>
<td>int</td>
<td>byte, short, int, char</td>
</tr>
<tr>
<td>long</td>
<td>byte, short, int, long, char</td>
</tr>
<tr>
<td>float</td>
<td>float, byte, short, int, long, char</td>
</tr>
<tr>
<td>double</td>
<td>float, double, byte, short, int, long, char</td>
</tr>
<tr>
<td>boolean</td>
<td>boolean</td>
</tr>
<tr>
<td>char</td>
<td>char</td>
</tr>
</tbody>
</table>

Sample Assignments

- This is a valid assignment:
  ```
  float salesTax = .05f;
  double taxRate = salesTax;
  ```

- This is invalid because the `float` data type is lower in precision than the `double` data type:
  ```
  double taxRate = .05;
  float salesTax = taxRate;
  ```

String Literals

- `String` is actually a class, not a basic data type; `String` variables are objects
- `String` literal: text contained within double quotes
- Example of `String` literals:
  ```
  "Hello"
  "Hello world"
  "The value of x is "
  ```

String Concatenation Operator (+)

- Combines `String` literals with other data types for printing
- Example:
  ```
  String hello = "Hello";
  String there = "there";
  String greeting = hello + ' ' + there;
  System.out.println( greeting );
  ```
  
  Output is:
  ```
  Hello there
  ```

Common Error Trap

- `String` literals must start and end on the same line. This statement:
  ```
  System.out.println( "Never pass a water fountain without taking a drink" );
  ```
  generates these compiler errors:
  ```
  unclosed string literal
  ')' expected
  ```
- Break long `Strings` into shorter `Strings` and use the concatenation operator:
  ```
  System.out.println("Never pass a water fountain" + " without taking a drink");
  ```

Escape Sequences

- To include a special character in a `String`, use an escape sequence

<table>
<thead>
<tr>
<th>Character</th>
<th>Escape Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Newline</td>
<td>\n</td>
</tr>
<tr>
<td>Tab</td>
<td>\t</td>
</tr>
<tr>
<td>Double quotes</td>
<td>&quot;</td>
</tr>
<tr>
<td>Single quote</td>
<td>'</td>
</tr>
<tr>
<td>Backslash</td>
<td>\</td>
</tr>
<tr>
<td>Backspace</td>
<td>\b</td>
</tr>
<tr>
<td>Carriage return</td>
<td>\r</td>
</tr>
<tr>
<td>Form feed</td>
<td>\f</td>
</tr>
</tbody>
</table>

See Example 2.3 Literals.java
Declare a variable only once
Once a variable is declared, its data type cannot be changed.
These statements:
  `double twoCents;
  double twoCents = .02;`
generate this compiler error:
  `twoCents is already defined`

Once a variable is declared, its data type cannot be changed.
These statements:
  `double cashInHand;
  int cashInHand;`
generate this compiler error:
  `cashInHand is already defined`

**Constants**
- Value cannot change during program execution
- Syntax:
  
  ```java
  final dataType constantIdentifier = assignedValue;
  ```
  
  Note: assigning a value when the constant is declared is optional. But a value must be assigned before the constant is used.

  See Example 2.4 Constants.java

- Use all capital letters for constants and separate words with an underscore:
  Example:
  ```java
  final double TAX_RATE = .05;
  ```

- Declare constants at the top of the program so their values can easily be seen
- Declare as a constant any data that should not change during program execution

**Expressions and Arithmetic Operators**
- The Assignment Operator and Expressions
- Arithmetic Operators
- Operator Precedence
- Integer Division and Modulus
- Division by Zero
- Mixed-Type Arithmetic and Type Casting
- Shortcut Operators

**Assignment Operator**

Syntax:

```java
  target = expression;
```

expression: operators and operands that evaluate to a single value

--value is then assigned to target
--target must be a variable (or constant)
--value must be compatible with target's data type
Examples: Assignment

```java
int numPlayers = 10; // numPlayers holds 10
numPlayers = 8; // numPlayers now holds 8

int legalAge = 18;
int voterAge = legalAge;
```

The next statement is illegal
```java
int height = weight * 2; // weight is not defined
```
and generates the following compiler error:
```java
illegal forward reference
```

Arithmetic Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>addition</td>
</tr>
<tr>
<td>-</td>
<td>subtraction</td>
</tr>
<tr>
<td>*</td>
<td>multiplication</td>
</tr>
<tr>
<td>/</td>
<td>division</td>
</tr>
<tr>
<td>%</td>
<td>modulus (remainder after division)</td>
</tr>
</tbody>
</table>

Operator Precedence

<table>
<thead>
<tr>
<th>Operator</th>
<th>Order of evaluation</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>( )</td>
<td>left - right</td>
<td>parenthesis for explicit grouping</td>
</tr>
<tr>
<td>* / %</td>
<td>left - right</td>
<td>multiplication, division, modulus</td>
</tr>
<tr>
<td>+ -</td>
<td>left - right</td>
<td>addition, subtraction</td>
</tr>
<tr>
<td>=</td>
<td>right - left</td>
<td>assignment</td>
</tr>
</tbody>
</table>

Example

- See Example 2.7 SimpleOperators.java Page 65

Example

You have 2 quarters, 3 dimes, and 2 nickels. How many pennies are these coins worth?

```java
int pennies = 2 * 25 + 3 * 10 + 2 * 5;
    =   50   +   30   + 10
    =  90
```

Another Example

Translate \( \frac{x}{2y} \) into Java:

```java
// incorrect!
double result = x / 2 * y;
=> x * y
    2
// correct
double result = x / ( 2 * y );
```
**Integer Division & Modulus**
- When dividing two integers:
  - the quotient is an integer
  - the remainder is truncated (discarded)
- To get the remainder, use the modulus operator with the same operands
- See Example 2.8 DivisionAndModulus.java

**Division by Zero**
- Integer division by 0:
  - Example: int result = 4 / 0;
  - No compiler error, but at run time, JVM generates ArithmeticException and program stops executing
  - Floating-point division by 0:
    - If dividend is not 0, the result is Infinity
    - If dividend and divisor are both 0, the result is NaN (not a number)
- See Example 2.9 DivisionByZero.java

**Mixed-Type Arithmetic**
- When performing calculations with operands of different data types:
  - Lower-precision operands are promoted to higher-precision data types, then the operation is performed
  - Promotion is effective only for expression evaluation; not a permanent change
  - Called "implicit type casting"
- Bottom line: any expression involving a floating-point operand will have a floating-point result.

**Rules of Promotion**
Applies the first of these rules that fits:
1. If either operand is a double, the other operand is converted to a double.
2. If either operand is a float, the other operand is converted to a float.
3. If either operand is a long, the other operand is converted to a long.
4. If either operand is an int, the other operand is promoted to an int.
5. If neither operand is a double, float, long, or an int, both operands are promoted to int.

**Explicit Type Casting**
- Syntax:
  
  `(dataType)( expression )`

  Note: parentheses around expression are optional if expression consists of 1 variable
- Useful for calculating averages
- See Example 2.10, MixedDataTypes.java

**Shortcut Operators**
- `++` increment by 1
- `--` decrement by 1
- Example:
  
  `count++;` // count = count + 1;
  `count--;` // count = count - 1;

  Postfix version (`var++`, `var--`): use value of `var` in expression, then increment or decrement.
  Prefix version (`++var`, `--var`): increment or decrement `var`, then use value in expression
- See Example 2.11 ShortcutOperators
### More Shortcut Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Example</th>
<th>Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>+=</td>
<td>a += 3;</td>
<td>a = a + 3;</td>
</tr>
<tr>
<td>-=</td>
<td>a -= 10;</td>
<td>a = a - 10;</td>
</tr>
<tr>
<td>*=</td>
<td>a *= 4;</td>
<td>a = a * 4;</td>
</tr>
<tr>
<td>/=</td>
<td>a /= 7;</td>
<td>a = a / 7;</td>
</tr>
<tr>
<td>%=</td>
<td>a %= 10;</td>
<td>a = a % 10;</td>
</tr>
</tbody>
</table>

### Common Error Trap

- No spaces are allowed between the arithmetic operator and the equals sign
- Note that the correct sequence is `+=`, not `=+`  
  
  Example: add 2 to a  
  
  // incorrect  
  a =+ 2; // a = +2; assigns 2 to 2  
  
  // correct  
  a += 2; // a = a + 2;  

### Operator Precedence

<table>
<thead>
<tr>
<th>Operator</th>
<th>Order of evaluation</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>( )</td>
<td>left - right</td>
<td>parenthesis for explicit grouping</td>
</tr>
<tr>
<td>++ --</td>
<td>right - left</td>
<td>preincrement, predecrement</td>
</tr>
<tr>
<td>++ --</td>
<td>right - left</td>
<td>postincrement, postdecrement</td>
</tr>
<tr>
<td>* / %</td>
<td>left - right</td>
<td>multiplication, division, modulus</td>
</tr>
<tr>
<td>+ -</td>
<td>left - right</td>
<td>addition or String</td>
</tr>
<tr>
<td>+= -= *= /= %=</td>
<td>right - left</td>
<td>assignment</td>
</tr>
</tbody>
</table>