Chapter 2

Primitive Types and Simple I/O

- Primitive Data types
- Strings: a class
- Assignment
- Expressions
- Keyboard and Screen I/O
- Documentation & Style
What is a program variable?

- A named location to store data
  - a container for data
- It can hold only one type of data
  - for example only integers, only floating point (real) numbers, or only characters
Creating Variables

- All program variables **must** be *declared* before using them.

- A variable declaration associates a name with a storage location in memory and specifies the type of data it will store:

  ```
  Type Variable_1, Variable_2, ...;
  ```

- For example, to create three integer variables to store the number of baskets, number of eggs per basket, and total number of eggs:

  ```
  int numberOfBaskets, eggsPerBasket, totalEggs;
  ```
Changing the Value of a Variable

Usually a variable is changed (assigned a different value) somewhere in the program

- May be calculated from other values:
  
  \[
  \text{totalEggs} = \text{numberOfBaskets} \times \text{eggsPerBasket};
  \]

- or read from keyboard input:

  \[
  \text{totalEggs} = \text{SavitchIn.readLineInt}();
  \]
Two Main Kinds of *Types* in Java

### Primitive data types
- The simplest types
- Cannot decompose into other types
- Values only, no methods
- Examples:
  - `int` - integer
  - `double` - floating point (real)
  - `char` - character

### Class types
- More complex
- Composed of other types (primitive or class types)
- Both data and methods
- Examples:
  - `SavitchIn`
  - `String`
Identifiers

- An identifier is the name of something (e.g. a variable, object, or method) used in a Java program.

- Syntax rules for identifiers tell what names are allowed.

- Naming conventions are not required by the compiler but are good practice.
Syntax Rules for Identifiers

**Identifiers**

- cannot be reserved words (e.g. “if,” “for”, etc.– see App. 1)
- must contain only letters, digits, and the underscore character, `_`
- cannot have a digit for the first character.
  - `$` is allowed but has special meaning, so do not use it.
- have no official length limit (there is always a finite limit, but it is very large and big enough for reasonable names)
- **are case sensitive!**
  - `junk`, `JUNK`, and `Junk` are three valid and different identifiers, so be sure to be careful in your typing!
- Note that no spaces or dots are allowed.
Naming Conventions

- Always use meaningful names, e.g. `finalExamScore`, instead of something like `x`, or even just `score`.
- Use only letters and digits.
- Capitalize interior words in multi-word names, e.g. `answerLetter`.
- Names of classes start with an uppercase letter.  
  » *every program in Java is a class as well as a program.*
- Names of variables, objects, and methods start with a lowercase letter.
Primitive Numeric Data Types

- integer—whole number
  examples: 0, 1, -1, 497, -6902
  » four data types: byte, short, int, long

- floating-point number—includes fractional part
  examples: 9.99, 3.14159, -5.63, 5.0
  » Note: 5.0 is a floating-point number even though the fractional part happens to be zero.
  » two data types: float, double
The `char` Data Type

- The `char` data type stores a single “printable” character

- For example:
  ```java
  char answer = 'y';
  System.out.println(answer);
  ```
  prints (displays) the letter `y`
# Primitive Data Types

<table>
<thead>
<tr>
<th>Type Name</th>
<th>Kind of Value</th>
<th>Memory Used</th>
<th>Size Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>integer</td>
<td>1 byte</td>
<td>-128 to 127</td>
</tr>
<tr>
<td>short</td>
<td>integer</td>
<td>2 bytes</td>
<td>-32768 to 32767</td>
</tr>
<tr>
<td>int</td>
<td>integer</td>
<td>4 bytes</td>
<td>-2,147,483,648 to 2,147,483,647</td>
</tr>
<tr>
<td>long</td>
<td>integer</td>
<td>8 bytes</td>
<td>-9,223,372,036,854,775,808 to 9,223,374,036,854,775,808</td>
</tr>
<tr>
<td>float</td>
<td>floating point</td>
<td>4 bytes</td>
<td>+/- 3.4028… x 10^{38} to +/- 1.4023… x 10^{-45}</td>
</tr>
<tr>
<td>double</td>
<td>floating point</td>
<td>8 bytes</td>
<td>+/- 1.767… x 10^{308} to +/- 4.940… x 10^{-324}</td>
</tr>
<tr>
<td>char</td>
<td>single character (Unicode)</td>
<td>2 bytes</td>
<td>all Unicode characters</td>
</tr>
<tr>
<td>boolean</td>
<td><em>true</em> or <em>false</em></td>
<td>1 bit</td>
<td>not applicable</td>
</tr>
</tbody>
</table>
Which Ones to Know for Now

Display in text is for reference; for now stick to these simple primitive types:

- **int**
  - just whole numbers
  - may be positive or negative
  - no decimal point

- **char**
  - just a single character
  - uses *single* quotes
  - for example
    ```java
    char letterGrade = `A``;``
  ```

- **double**
  - real numbers, both positive and negative
  - has a decimal point
    (fractional part)
  - two formats
    - number with decimal point, e.g. 514.061
    - `e` (or *scientific*, or *floating-point*) notation,
      e.g. 5.14061 e2, which means 5.14061 \times 10^2
Assignment Statements

- most straightforward way to change value of a variable

\[ \text{Variable} = \text{Expression} \]

\[ \text{answer} = 42; \]

- = is assignment operator

- evaluate expression on right-hand side of the assignment operator

- variable on the left-hand side of the assignment operator gets expression value as new value
Assignment Operator  =

- The assignment operator is not the same as the equals sign in algebra.
- It means -
  “Assign the value of the expression on the right side to the variable on the left side.”
- Can have the same variable on both sides of the assignment operator:
  ```java
  int count = 10; // initialize counter to ten
  count = count - 1; // decrement counter
  new value of count = 10 - 1 = 9
  ```
Specialized Assignment Operators

- A shorthand notation for performing an operation on and assigning a new value to a variable
- General form: `var <op>= expression;`
  - equivalent to: `var = var <op> (expression);`
  - `<op>` is +, -, *, /, or %
- Examples:
  ```java
  amount += 5;
  //amount = amount + 5;
  
  amount *= 1 + interestRate;
  //amount = amount * (1 + interestRate);
  ```
- Note that the right side is treated as a unit (put parentheses around the entire expression)
Returned Value

- Expressions return values: the number produced by an expression is “returned”, i.e. it is the “return value.”
  
  ```java
  int numberOfBaskets, eggsPerBasket, totalEggs;
  numberOfBaskets = 5;
  eggsPerBasket = 8;
  totalEggs = numberOfBaskets * eggsPerBasket;
  ```

  » in the last line `numberOfBaskets` returns the value 5 and `eggsPerBasket` returns the value 8

  » `numberOfBaskets * eggsPerBasket` is an expression that returns the integer value 40

- Similarly, methods return values
  
  ```java
  SavitchIn.readLine() is a method that returns a string read from the keyboard
  ```
Assignment Compatibility

- Can't put a square peg in a round hole
- Can't put a `double` value into an `int` variable
- In order to copy a value of one type to a variable of a different type, there must be a conversion.
- Converting a value from one type to another is called `casting`.
- Two kinds of casting:
  - automatic or implicit casting
  - explicit casting
Casting: changing the data type of the returned value

- Casting only changes the type of the returned value (the single instance where the cast is done), not the type of the variable.
- For example:
  ```java
  double x;
  int n = 5;
  x = n;
  ```
- Since `n` is an integer and `x` is a double, the value returned by `n` must be converted to type double before it is assigned to `x`
Implicit Casting

- Casting is done implicitly (automatically) when a “lower” type is assigned to a “higher” type.

- The data type hierarchy (from lowest to highest):
  
  byte ➔ short ➔ int ➔ long ➔ float ➔ double

- An `int` value will automatically be cast to a `double` value.
- A `double` value will not automatically be cast to an `int` value.
Implicit Casting Example: \textit{int} to \textit{double}

double \texttt{x};
int \texttt{n} = 5;
\texttt{x} = \texttt{n};

data type hierarchy:
\begin{center}
\begin{tabular}{c}
\texttt{byte} \rightarrow \texttt{short} \rightarrow \texttt{int} \rightarrow \texttt{long} \rightarrow \texttt{float} \rightarrow \texttt{double}
\end{tabular}
\end{center}

- the value returned by \texttt{n} is \textit{cast} to a double, then assigned to \texttt{x}
- \texttt{x} contains 5.000\ldots (as accurately as it can be encoded as a floating point number)
- This casting is done automatically because \texttt{int} is lower than \texttt{double} in the data type hierarchy
- The data type of the variable \texttt{n} is unchanged; is still an \texttt{int}
Data Types in an Expression: More Implicit Casting

- Some expressions have a mix of data types
- All values are automatically advanced (implicitly cast) to the highest level before the calculation
- For example:

```java
double a;
int n = 2;
float x = 5.1;
double y = 1.33;
a = (n * x)/y;
```

n and x are automatically cast to type double before performing the multiplication and division
Explicit Casting

- Explicit casting changes the data type of the value for a single use of the variable.
- Precede the variable name with the new data type in parentheses:
  
  `(data type) variableName`

  » The type is changed to `<data type>` only for the single use of the returned value where it is cast.

- For example:

  ```java
  int n;
  double x = 2.0;
  n = (int)x
  ```

  The value of `x` is converted from double to integer before assigning the value to `n`
Explicit casting is **required** to assign a higher type to a lower

- **ILLEGAL**: *Implicit* casting to a *lower* data type
  ```java
  int n;
  double x = 2.1;
  n = x;  //illegal in java
  ```
  It is illegal since `x` is double, `n` is an int, and double is a higher data type than integer

- **LEGAL**: *Explicit* casting to a lower data type
  ```java
  int n;
  double x = 2.1;
  n = (int)x;  //legal in java
  ```

You can always use an explicit cast where an implicit one will be done automatically, but it is not necessary
Truncation When Casting a `double` to an Integer

- Converting (casting) a double to integer does **not** round; it *truncates*
  - the fractional part is lost (discarded, ignored, thrown away)
- For example:
  ```java
  int n;
  double x = 2.99999;
  n = (int)x;
  ```
  - the value of `n` is now 2 (truncated value of `x`)
  - the cast is required
- This behavior is useful for some calculations, as demonstrated in *Case Study: Vending Machine Change*
Characters as Integers

- Characters are actually stored as integers according to a special code
  - each printable character (letter, number, punctuation mark, space, and tab) is assigned a different integer code
  - the codes are different for upper and lower case
  - for example 97 may be the integer value for ‘a’ and 65 for ‘A’
- ASCII (Appendix 3) and Unicode are common character codes
- Unicode includes all the ASCII codes plus additional ones for languages with an alphabet other than English
- Java uses Unicode
Casting a `char` to an `int`

- Casting a char value to int produces the ASCII/Unicode value
- For example, what would the following display?
  ```java
  char answer = `y`;
  System.out.println(answer);
  System.out.println((int)answer);
  ```

- Answer: the letter ‘y’ on one line followed by the ASCII code for ‘y’ (lower case) on the next line:
  ```
  >y
  >89
  >
  ```
Assigning Initial Values to Variables

- Initial values *may or may not* be assigned when variables are declared:

```java
//These are not initialized when declared
//and have unknown values
int totalEggs, numberOfBaskets, eggsPerBasket;

//These are initialized to 0 when declared
int totalEggs = 0;
int numberOfBaskets = 0;
int eggsPerBasket = 0;
```

- Programming tip: it is good programming practice always to initialize variables.
GOTCHA: Imprecision of Floating Point Numbers

- Computers store numbers using a fixed number of bits, so not every real (floating point) number can be encoded precisely.
  - An infinite number of bits would be required to precisely represent any real number.
- For example, if a computer can represent up to 10 decimal digits, the number 2.5 may be stored as 2.499999999 if that is the closest it can come to 2.5.
- Integers, on the other hand, are encoded precisely.
  - If the value 2 is assigned to an int variable, its value is precisely 2.
- This is important in programming situations you will see later in the course.
Arithmetic Operators

- addition (+), subtraction (-), multiplication (*), division (/)
- can be performed with numbers of any integer type, floating-point type, or combination of types
- result will be the highest type that is in the expression
- Example:
  
  \[
  \text{amount} - \text{adjustment}
  \]
  
  • result will be int if both \text{amount} and \text{adjustment} are int
  • result will be float if amount is int and adjustment is float

Data type hierarchy: byte ➔ short ➔ int ➔ long ➔ float ➔ double
Truncation When Doing Integer Division

- No truncation occurs if at least one of the values in a division is type `float` or `double` (all values are promoted to the highest data type).

- Truncation occurs if all the values in a division are integers.

- For example:

```java
int a = 4, b = 5, c;
double x = 1.5, y;
y = b/x; // value returned by b is cast to double
    // value of y is approximately 3.33333
   c = b/a; // all values are ints so the division
   // truncates: the value of c is 1!
```
The Modulo Operator:  \( a \% b \)

- Used with integer types
- Returns the remainder of the division of \( b \) by \( a \)
- For example:

```java
int a = 57; b = 16, c;
c = a % b;
```

\( c \) now has the value 9, the remainder when 57 is divided by 16

- A very useful operation: see *Case Study: Vending Machine Change*
Arithmetic Operator Precedence and Parentheses

- Java expressions follow rules similar to real-number algebra.

- Use parentheses to force precedence.

- Do not clutter expressions with parentheses when the precedence is correct and obvious.
Examples of Expressions

<table>
<thead>
<tr>
<th>Ordinary Math Expression</th>
<th>Java Expression (preferred form)</th>
<th>Java Fully Parenthesized Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>rate² + delta</td>
<td>rate*rate + delta</td>
<td>(rate*rate) + delta</td>
</tr>
<tr>
<td>2(salary + bonus)</td>
<td>2 * (salary + bonus)</td>
<td>2 * (salary + bonus)</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>time + 3 * mass</td>
<td>1/(time + 3 * mass)</td>
<td>1/(time + (3 * mass))</td>
</tr>
<tr>
<td>a - 7</td>
<td>(a - 7) / (t + 9 * v)</td>
<td>(a - 7) / (t + (9 * v))</td>
</tr>
</tbody>
</table>
Vending Machine Change

Excerpt from the ChangeMaker.java program:

```java
int amount, originalAmount,
    quarters, dimes, nickels, pennies;

originalAmount = amount;
quarters = amount/25;
amount = amount%25;
dimes = amount/10;
amount = amount%10;
nickels = amount/5;
amount = amount%5;
pennies = amount;
```

If `amount` is 90 then 90/25 will be 3, so there are three quarters.

If `amount` is 90 then the remainder of 90/25 will be 15, so 15 cents change is made up of other coins.
Shorthand notation for common arithmetic operations on variables used for counting

- Some counters count up, some count down, but they are integer variables
- The counter can be incremented (or decremented) before or after using its current value

```java
int count;
...
++count  // preincrement count: count = count + 1 before using it
count++  // postincrement count: count = count + 1 after using it
--count  // predecrement count: count = count - 1 before using it
count--  // postdecrement count: count = count - 1 after using it
```
Increment and Decrement Operator Examples

common code
int n = 3;
int m = 4;
int result;

What will be the value of m and result after each of these executes?
(a) result = n * ++m; // preincrement m
(b) result = n * m++; // postincrement m
(c) result = n * --m; // prededrement m
(d) result = n * m--; // postdecrement m
Answers to Increment/Decrement Operator Questions

(a) 1) \( m = m + 1; \) // \( m = 4 + 1 = 5 \)
    2) \( \text{result} = n * m; \) // \( \text{result} = 3 * 5 = 15 \)

(b) 1) \( \text{result} = n * m; \) // \( \text{result} = 3 * 4 = 12 \)
    2) \( m = m + 1; \) // \( m = 4 + 1 = 5 \)

(c) 1) \( m = m - 1; \) // \( m = 4 - 1 = 3 \)
    2) \( \text{result} = n * m; \) // \( \text{result} = 3 * 3 = 9 \)

(b) 1) \( \text{result} = n * m; \) // \( \text{result} = 3 * 4 = 12 \)
    2) \( m = m - 1; \) // \( m = 4 - 1 = 3 \)
The *String* Class

- A string is a sequence of characters
- The String class is used to store strings
- The String class has methods to operate on strings
- String constant: one or more characters in *double* quotes
- Examples:

```java
char charVariable = `a`; // single quotes
String stringVariable = "a"; // double quotes
String sentence = "Hello, world";
```
String Variables

- Declare a String variable:
  ```java
  String greeting;
  ```

- Assign a value to the variable
  ```java
  greeting = "Hello!";
  ```

- Use the variable as a String argument in a method:
  ```java
  System.out.println(greeting);
  ```
  causes the string Hello! to be displayed on the screen
Concatenating (Appending) Strings

Stringing together strings - the “+” operator for Strings:

```java
String name = "Mondo";
String greeting = "Hi, there!";
System.out.println(greeting + name + "Welcome");
```

causes the following to display on the screen:

>Hi, there!MondoWelcome

Note that you have to remember to include spaces if you want it to look right:

```java
System.out.println(greeting + " " + name + " Welcome");
```

causes the following to display on the screen:

>Hi, there! Mondo Welcome

>`
Indexing Characters within a String

- The index of a character within a string is an integer starting at 0 for the first character and gives the position of the character.
- The `charAt(Position)` method returns the char at the specified position.
- The `substring(Start, End)` method returns the string from position `Start` to position `End-1`.
- For example:
  ```java
  String greeting = "Hi, there!";
  greeting.charAt(0) returns 'H'
  greeting.charAt(2) returns ','
  greeting.substring(4,7) returns "the"
  ```

<table>
<thead>
<tr>
<th></th>
<th>H</th>
<th>i</th>
<th>,</th>
<th>t</th>
<th>h</th>
<th>e</th>
<th>r</th>
<th>e</th>
<th>!</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>
Escape Characters

- How do you print characters that have special meaning? For example, how do you print the following string?
  The word "hard"

  Would this do it?
  System.out.println("The word "hard"");
  No, it would give a compiler error - it sees the string The word between the first set of double quotes and is confused by what comes after

- Use the backslash character, \", to escape the special meaning of the internal double quotes:
  System.out.println("The word \"hard\""); //this works
More Escape Characters

Use the following escape characters to include the character listed in a quoted string:

\"  Double quote.
\\  Single quote.
\\\  Backslash.
\n  New line. Go to the beginning of the next line.
\r  carriage return. Go to the beginning of the current line.
\t  Tab. White space up to the next tab stop.
Screen Output: print and println

- Sometimes you want to print part of a line and not go to the next line when you print again.
- Two methods, one that goes to a new line and one that does not:
  System.out.println(…);//ends with a new line
  System.out.print(…);//stays on the same line.
- For example:
  System.out.print("This will all ");
  System.out.println("appear on one line");
- System.out.print() works similar to the "+" operator:
  System.out.println("This will all "+ "appear on one line, too");
Program I/O

- I/O - Input/Output
- Keyboard is the normal input device
- Screen is the normal output device
- Classes are used for I/O
- They are generally add-on classes (not actually part of Java)
- Some I/O classes are always provided with Java, others are not
I/O Classes

- We have been using an output method from a class that automatically comes with Java:
  ```java
  System.out.println()
  ```
- But Java does not automatically have an input class, so one must be added
  » SavitchIn is a class specially written to do keyboard input
- SavitchIn.java is provided with the text - see Appendix 4
- Examples of SavitchIn methods for keyboard input:
  ```java
  readLineInt()
  readLineDouble()
  readLineNonwhiteChar()
  ```
- Gotcha: remember Java is case sensitive, for example `readLineNonWhiteChar()` will not work
int amount, originalAmount,
   quarters, dimes, nickels, pennies;
System.out.println("Enter a whole number...");
System.out.println("I will output ... coins");
System.out.println("that equals that amount ...");

amount = SavitchIn.readLineInt();
originalAmount = amount;
Keyboard Input Gotchas

Note the two variations for reading each type of number

**readLine variation**
- reads a whole line
- asks the user to reenter if it is not the right format
- Try to use these
- Examples:
  - readLineInt()
  - readLineDouble()

**read variation**
- reads just the number
- aborts the program if it is not the right format
- Avoid using these
- Examples:
  - readInt()
  - readDouble()
User-Friendly Input

- Print a prompt so that the user knows what kind of information is expected.
- Echo the information that the user typed in so that it can be verified.

```java
System.out.println("Enter the number of trolls:");
int trolls = SavitchIn.readLineInt();
System.out.println(trolls + " trolls");
```

Sample output with user input in italic:

Enter the number of trolls: 
38
38 trolls
A little practical matter:  
If the screen goes away too quickly …

If the output (screen display) of your programs does not stay on the screen, use this code:

```java
System.out.println(“Press any key to end program.”);
String junk;
junk = SavitchIn.readLine();
```

- The display stops until you enter something
- Whatever you enter is stored in variable `junk` but is never used
  - it is “thrown away”
Documentation and Style

- Use meaningful names for variables, classes, etc.
- Use indentation and line spacing as shown in the examples in the text.
- Always include a “prologue” (an brief explanation of the program at the beginning of the file).
- Use all lower case for variables, except capitalize internal words (eggsPerBasket).
- Use all upper case for variables that have a constant value, PI for the value of pi (3.14159…) (see text for more examples).
Comments

- *Comment*—text in a program that the compiler ignores
- Does not change what the program does, only explains the program
- Write meaningful and useful comments
- Comment the *non*-obvious
- Assume a *reasonably* knowledgeable reader
- // for single-line comments
- /* … */ for multi-line comments
Named Constants

- **Named constant**—using a name instead of a value
- Example: use `MORTGAGE_INTEREST_RATE` instead of 8.5

- Advantages of using named constants
  - Easier to understand program because reader can tell how the value is being used
  - Easier to modify program because value can be changed in one place (the definition) instead of being changed everywhere in the program.
  - Avoids mistake of changing same value used for a different purpose
Defining Named Constants

```java
public static final double PI = 3.14159;
```

public—no restrictions on where this name can be used
static—must be included, but explanation has to wait
final—the program is not allowed to change the value

- The remainder of the definition is similar to a variable declaration and gives the type, name, and initial value.
- A declaration like this is usually at the beginning of the file and is not inside the main method definition.
Summary
Part 1

- Variables hold values and have a *type*
  - The type of a Java variable is either a primitive type or a class
  - Common primitive types in Java include `int`, `double`, and `char`
  - A common class type in Java is `String`
  - Variables must be declared

- Parentheses in arithmetic expressions ensure correct execution order

- Use `SavitchIn` methods for keyboard input
  - `SavitchIn` is not part of standard Java
Summary
Part 2

- Good programming practice:
  » Use meaningful names for variables
  » Initialize variables
  » Use variable names (in upper case) for constants
  » Use comments sparingly but wisely, e.g. to explain non-obvious code
  » Output a prompt when the user is expected to enter data from the keyboard
  » Echo the data entered by the user