Chapter 2: Data and Expressions
CS 121

Department of Computer Science
College of Engineering
Boise State University

January 15, 2015
Chapter 2

- Part 1: Data Types
  Go to part 1

- Part 2: Expressions and Scanner
  Go to part 2
Part 1: Data Types

- Character Strings
  - Concatenation
  - Escape Sequences
- Java Primitive Types
- Declaring and Using Variables

Go to index.
Character Strings

- A sequence of characters can be represented as a string literal by putting double quotes around it.
- “This is a string literal.” “So is this.”
- A character string is an object in Java, defined by the String class.
- Every string literal represents a String object.
A sequence of characters can be represented as a string literal by putting double quotes around it.

“This is a string literal.” “So is this.”

A character string is an object in Java, defined by the String class.

Every string literal represents a String object.

See http://docs.oracle.com/javase/8/docs/api/java/lang/String.html.
Printing Strings

- The `System.out` object represents a destination (the monitor) to which we can send output.
- We can invoke the `println` and `print` methods of the `System.out` object to print a character string.
Printing Strings

- The `System.out` object represents a destination (the monitor) to which we can send output.
- We can invoke the `println` and `print` methods of the `System.out` object to print a character string.
  - `println` – prints a new line character (`'\n'`) after the string.
  - `print` – does NOT print a new line character (`'\n'`) after the string.
- **Example:** `Countdown.java`
The string concatenation operator (+) appends one string to the end of another.

"Peanut butter " + "and jelly"

Allows strings to be broken across multiple lines.

"If this was a long string, we may want it on " + "two lines so we can see it all in our editor"

Also used to append numbers to a string.

"We will have " + 10 + " quizzes this semester."
The + operator is also used for addition.
The function it performs depends on the context.
  - **String concatenation**
    - Both operands are strings.
    - One operand is a string and one is a number.
  - **Addition**
    - Both operands are numeric.
  - **Example:** Addition.java

Precedence: evaluated **left to right**, but can use parenthesis to force order (more about this later).
Escape Sequences

What if we wanted to actually print the “ character??

```
System.out.println("I said "Hello" to you");
```

Our compiler is confused! Do you know why?

We can fix it with an escape sequence – a series of characters that represents a special character.

Begins with a backslash character (\).
What if we wanted to actually print the " character??
Let’s try it.

```java
System.out.println("I said "Hello" to you");
```
Escape Sequences

- What if we wanted to actually print the " character??
- Let's try it.
  ```java
  System.out.println("I said "Hello" to you");
  ```
- Our compiler is confused! Do you know why?
What if we wanted to actually print the " character??
Let’s try it.

System.out.println("I said "Hello" to you");

Our compiler is confused! Do you know why?

We can fix it with an escape sequence – a series of characters that represents a special character.
What if we wanted to actually print the " character??
Let's try it.
System.out.println("I said "Hello" to you");
Our compiler is confused! Do you know why?
We can fix it with an escape sequence – a series of characters that represents a special character.
Begins with a backslash character (\).
System.out.println("I said \"Hello\" to you");
### Some Java Escape Sequences

<table>
<thead>
<tr>
<th>Escape Sequence</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>\</td>
<td>backspace</td>
</tr>
<tr>
<td>\t</td>
<td>tab</td>
</tr>
<tr>
<td>\n</td>
<td>newline</td>
</tr>
<tr>
<td>\r</td>
<td>carriage return</td>
</tr>
<tr>
<td>&quot;</td>
<td>double quote</td>
</tr>
<tr>
<td>'</td>
<td>single quote</td>
</tr>
<tr>
<td>\</td>
<td>backslash</td>
</tr>
</tbody>
</table>
Using Java Escape Sequences

Example: Roses.java
Using Java Escape Sequences

- **Example:** Roses.java
- **Example:** CarriageReturnDemo.java (must run from command-line)
Primitive Data Types

- There are 8 primitive data types in Java (varies in other languages)
- Integers
  - `byte`, `short`, `int`, `long`
- Floating points (decimals)
  - `float`, `double`
- Characters
  - `char`
- Boolean values (true/false)
  - `boolean`
## Numeric Types

<table>
<thead>
<tr>
<th>Type</th>
<th>Storage</th>
<th>Min Value</th>
<th>Max Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>8 bits</td>
<td>-128</td>
<td>127</td>
</tr>
<tr>
<td>short</td>
<td>16 bits</td>
<td>-32,768</td>
<td>32,767</td>
</tr>
<tr>
<td>int</td>
<td>32 bits</td>
<td>-2,147,483,648</td>
<td>2,147,483,647</td>
</tr>
<tr>
<td>long</td>
<td>64 bits</td>
<td>-9,223,372,036,854,775,808</td>
<td>9,223,372,036,854,775,807</td>
</tr>
<tr>
<td>float</td>
<td>32 bits</td>
<td>Approximately -3.4E+38 with 7 significant digits</td>
<td>Approximately 3.4E+38 with 7 significant digits</td>
</tr>
<tr>
<td>double</td>
<td>64 bits</td>
<td>Approximately -1.7E+308 with 15 significant digits</td>
<td>Approximately 1.7E+308 with 15 significant digits</td>
</tr>
</tbody>
</table>
A char stores a single character delimited by single quotes.

```java
char topGrade = 'A';
char comma = ',';
```

A char variable in Java can store any character from the Unicode character set.

- Each character corresponds to a unique 16-bit number.

We typically use characters from the ASCII character set.

- Older and smaller subset of Unicode (only 8-bits).

Unicode and ASCII tables available online.
Booleans

- Only two valid values for the `boolean` type.
- Reserved words `true` and `false`.
  ```java
  boolean done = false;
  ```
- Commonly used to represent two states (e.g. on/off)
Identifiers are words a programmer uses in a program.

- A-Z, a-z, 0-9, _, and $
- Can’t begin with digit.
- Case sensitive.
  - Total, total, and TOTAL are different
Identifiers are words a programmer uses in a program.
- A-Z, a-z, 0-9, _, and $
- Can’t begin with digit.
- Case sensitive.
  - Total, total, and TOTAL are different

Good practice to use different case style for different types of identifiers.
- title case for class names – Lincoln, HelloClass
- camel case for variables – count, nextCount
- upper case for constants – MAXIMUM, MINIMUM
Reserved Words

- **Reserved words** are special identifiers that have pre-defined meaning.
- They can’t be used in any other way.
- Some examples – public, static, void, class
- See page 7 in textbook for full list of words.
A variable is just a name for a location in memory.
A variable is just a name for a location in memory.

Variable names are identifiers. They must be unique.
A variable is just a name for a location in memory.

Variable names are identifiers. They must be unique.

Variables must be declared by specifying a name and the type of information it will hold.
A variable is just a name for a location in memory.
Variable names are identifiers. They must be unique.
Variables must be declared by specifying a name and the type of information it will hold.

```java
String name;
int radius, area, circumference;
```
A variable is just a name for a location in memory. Variable names are identifiers. They must be unique. Variables must be declared by specifying a name and the type of information it will hold.

```java
String name;
int radius, area, circumference;
```

When a variable is used in a program, the current value is used.
An assignment statement changes the value of a variable. The assignment operator is the equals sign (=).

\[
\text{radius} = 10;
\]
The value on the right-hand side is stored in the variable on the left.

Variables can also be initialized when they are declared.

\[
\text{int count} = 0;
\]
The previous value in \texttt{radius} is overwritten.

The type of the right-hand side must be compatible with the type of the variable.
The right-hand side can be an expression.

The expression will be evaluated \textit{first} and \textit{then} stored in the variable.

\begin{verbatim}
radius = 10;
radius = radius * 2; // double the radius
\end{verbatim}
The right-hand side can be an expression.
The expression will be evaluated \textit{first} and \textit{then} stored in the variable.

\begin{verbatim}
radius = 10;
radius = radius * 2; // double the radius
\end{verbatim}

What is the new value of \texttt{radius}?
The right-hand side can be an expression.

The expression will be evaluated \textit{first} and \textit{then} stored in the variable.

\begin{verbatim}
radius = 10;
radius = radius * 2; // double the radius
\end{verbatim}

What is the new value of \texttt{radius}? 20
A constant is an identifier (similar to a variable) that holds the same value during its entire existence.

It is constant, not variable.

The compiler will issue an error if you try to change the value of a constant.

In Java, we use the final modifier to declare a constant.

We typically use all caps to name constants.

```java
final int MAX_RADIUS = 1000;
```
Why do we need constants?

- Readability – give meaning to arbitrary literals.
- Program maintenance – only need to change value once.
- Program protection – establishes that a value should not change; less chance for error.
Exercises

- EX 2.2, EX 2.4, PP 2.1
Part 2: Expressions and Scanner

- Expressions
- Data conversions
- The Scanner class for interactive programs

Go to index.
Expressions

- An expression is a combination of one or more operators and operands.
- We focus on arithmetic expressions that produce numeric results.
Arithmetic expressions use the arithmetic operators.

- Addition: +
- Subtraction: -
- Multiplication: *
- Division: /
- Remainder (modulo): %
Arithmetic Expressions and Data Types

- If *any one* of the operands used by an arithmetic operator are *floating point* (*float* or *double*), then the result will be a floating point.

- For example:

  ```java
  int radius = 10;
  final double PI = 3.14159265358979323;
  double area = PI * radius * radius;
  ```

- If *both* operands used by an arithmetic operator are *floating point*, then the result will be a floating point.

- If *both* operands used by an arithmetic operator are *integer*, then the result will be an integer. *Be careful!!*
Division and Data Types

- If both operands of the division operator are integers, then the result will be an integer.
- This means we lose the fractional part of the result.
- For example, let’s assume we want to divide a wall into equal sections.

```java
int length = 15;
int sections = 2;
float newLength = length / sections;
```

- Let’s try this.
Division and Data Types

- If both operands of the division operator are integers, then the result will be an integer.
- This means we lose the fractional part of the result.
- For example, let’s assume we want to divide a wall into equal sections.

```java
int length = 15;
int sections = 2;
float newLength = length / sections;
```

- Let’s try this.
- How can we fix it?
Division and Data Types

- If both operands of the division operator are integers, then the result will be an integer.
- This means we lose the fractional part of the result.
- For example, let's assume we want to divide a wall into equal sections.
  
  ```java
  int length = 15;
  int sections = 2;
  float newLength = length / sections;
  ```

- Let's try this.

- How can we fix it?

- **Data conversion** – we’ll get to this soon.
Given two positive numbers, $a$ (the dividend) and $b$ (the divisor), $a \% n \ (a \mod n)$ is the remainder of the Euclidean division of $a$ by $n$.

\[
\begin{align*}
14 \div 3 &= 4 & 14 \% 3 &= 2 \\
8 \div 12 &= 0 & 8 \% 12 &= 8 \\
10 \div 2 &= 5 & 10 \% 2 &= 0 \\
7 \div 6 &= 1 & 7 \% 6 &= 1 \\
9 \div 0 &= \text{error} & 9 \% 0 &= \text{error}
\end{align*}
\]
Given two positive numbers, \( a \) (the dividend) and \( b \) (the divisor), \( a \% n \) (\( a \text{ mod } n \)) is the remainder of the Euclidean division of \( a \) by \( n \).

\[
\begin{align*}
14 \div 3 & = 4 & 14 \% 3 & = 2 \\
8 \div 12 & = 0 & 8 \% 12 & = 8
\end{align*}
\]
Remainder Operator (modulo)

Given two positive numbers, \( a \) (the dividend) and \( b \) (the divisor), \( a \% n \ (a \mod n) \) is the remainder of the Euclidean division of \( a \) by \( n \).

\[
\begin{align*}
14 / 3 & == 4 & 14 \% 3 & == 2 \\
8 / 12 & == 0 & 8 \% 12 & == 8 \\
10 / 2 & == & \\
\end{align*}
\]
Given two positive numbers, $a$ (the dividend) and $b$ (the divisor), $a \% n$ ($a \mod n$) is the remainder of the Euclidean division of $a$ by $n$.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>$14 \div 3$</td>
<td>$4$</td>
</tr>
<tr>
<td>$14 % 3$</td>
<td>$2$</td>
</tr>
<tr>
<td>$8 \div 12$</td>
<td>$0$</td>
</tr>
<tr>
<td>$8 % 12$</td>
<td>$8$</td>
</tr>
<tr>
<td>$10 \div 2$</td>
<td>$5$</td>
</tr>
<tr>
<td>$10 % 2$</td>
<td>$0$</td>
</tr>
<tr>
<td>$7 \div 6$</td>
<td>$1$</td>
</tr>
<tr>
<td>$7 % 6$</td>
<td>$1$</td>
</tr>
<tr>
<td>$9 \div 0$</td>
<td>Error</td>
</tr>
<tr>
<td>$9 % 0$</td>
<td>Error</td>
</tr>
</tbody>
</table>
Given two positive numbers, \( a \) (the dividend) and \( b \) (the divisor), \( a \mod n \) is the remainder of the Euclidean division of \( a \) by \( n \).

\[
\begin{align*}
14 & \div 3 \equiv 4 & 14 \mod 3 & \equiv 2 \\
8 & \div 12 \equiv 0 & 8 \mod 12 & \equiv 8 \\
10 & \div 2 \equiv 5 & 10 \mod 2 & \equiv 0 \\
7 & \div 6 & & \equiv
\end{align*}
\]
Given two positive numbers, $a$ (the dividend) and $b$ (the divisor), $a \mod b$ (or $a \text{ mod } b$) is the remainder of the Euclidean division of $a$ by $b$.

- $14 \div 3 = 4$ (with remainder 2)
- $8 \div 12 = 0$ (with remainder 8)
- $10 \div 2 = 5$ (with remainder 0)
- $7 \div 6 = 1$ (with remainder 1)
- $9 \div 0$ is undefined
Given two positive numbers, \( a \) (the dividend) and \( b \) (the divisor), \( a \% n \) (\( a \mod n \)) is the remainder of the Euclidean division of \( a \) by \( n \).

\[
\begin{array}{ccc}
14 \div 3 &=& 4 & 14 \% 3 &=& 2 \\
8 \div 12 &=& 0 & 8 \% 12 &=& 8 \\
10 \div 2 &=& 5 & 10 \% 2 &=& 0 \\
7 \div 6 &=& 1 & 7 \% 6 &=& 1 \\
9 \div 0 &=& \\
\end{array}
\]
Given two positive numbers, \( a \) (the dividend) and \( b \) (the divisor), \( a \mod n \) is the remainder of the Euclidean division of \( a \) by \( n \).

\[
\begin{align*}
14 \div 3 &= 4 & 14 \mod 3 &= 2 \\
8 \div 12 &= 0 & 8 \mod 12 &= 8 \\
10 \div 2 &= 5 & 10 \mod 2 &= 0 \\
7 \div 6 &= 1 & 7 \mod 6 &= 1 \\
9 \div 0 &= \text{error} & 9 \mod 0 &= \text{error}
\end{align*}
\]
Given two positive numbers, \( a \) (the dividend) and \( b \) (the divisor), \( a \% n \) (\( a \pmod{n} \)) is the remainder of the Euclidean division of \( a \) by \( n \).

\[
\begin{align*}
14 & / 3 \quad == \quad 4 & 14 & \% 3 \quad == \quad 2 \\
8 & / 12 \quad == \quad 0 & 8 & \% 12 \quad == \quad 8 \\
10 & / 2 \quad == \quad 5 & 10 & \% 2 \quad == \quad 0 \\
7 & / 6 \quad == \quad 1 & 7 & \% 6 \quad == \quad 1 \\
9 & / 0 \quad == \quad error & 9 & \% 0 \quad == \quad error
\end{align*}
\]
Remainder Operator (modulo)

- Typically used to determine if a number is odd or even.
- How?
Just like in math, operators can be combined into complex expressions.

\[
\text{result} = \text{total} + \frac{\text{count}}{\text{max}} - \text{offset};
\]
Operator Precedence (Order of Operations)

- Just like in math, operators can be combined into complex expressions.
  \[ \text{result} = \text{total} + \frac{\text{count}}{\text{max}} - \text{offset}; \]

- Operators have well-defined *precedence* to determine order of evaluation.
  \[ \text{result} = \text{total} + \frac{\text{count}}{\text{max}} - \text{offset}; \]
  \[ 4 \quad 2 \quad 1 \quad 3 \]
## Operator Precedence

<table>
<thead>
<tr>
<th>Precedence</th>
<th>Operator</th>
<th>Operation</th>
<th>Association</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>()</td>
<td>parenthesis</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>+</td>
<td>unary plus</td>
<td>R to L</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>unary minus</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>*</td>
<td>multiplication</td>
<td>L to R</td>
</tr>
<tr>
<td></td>
<td>/</td>
<td>division</td>
<td></td>
</tr>
<tr>
<td></td>
<td>%</td>
<td>modulo</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>+</td>
<td>addition</td>
<td>L to R</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>subtraction</td>
<td></td>
</tr>
<tr>
<td></td>
<td>+</td>
<td>string concatenation</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>=</td>
<td>assignment</td>
<td>R to L</td>
</tr>
</tbody>
</table>
In-Class Exercise

Determine the order of evaluation in the following expressions.

1) \( a + b + c + d + e \)

2) \( a + b * c - d / e \)

3) \( a / (b + c) - d \% e \)

4) \( a / (b * (c + (d - e))) \)
Determine the order of evaluation in the following expressions.

1) \( a + b + c + d + e \)
In-Class Exercise

Determine the order of evaluation in the following expressions.

1) \( a + b + c + d + e \)
   
   1 2 3 4

2) \( a + b \times c - d \div e \)
   
   3 1 4 2

3) \( a \div (b + c) - d \% e \)
   
   2 1 4 3

4) \( a \div (b \times (c + (d - e))) \)
   
   4 3 2 1
In-Class Exercise

- Determine the order of evaluation in the following expressions.

1) \( a + b + c + d + e \)
   1 2 3 4

2) \( a + b \times c - d / e \)

Chapter 2: Data and Expressions
In-Class Exercise

- Determine the order of evaluation in the following expressions.

1) \( a + b + c + d + e \)
   
   1 2 3 4

2) \( a + b * c - d / e \)
   
   3 1 4 2

3) \( a / (b + c) - d \% e \)
   
   2 1 4 3

4) \( a / (b * (c + (d - e))) \)
   
   4 3 2 1
Determine the order of evaluation in the following expressions.

1) \[a + b + c + d + e\]
   \[1 \ 2 \ 3 \ 4\]

2) \[a + b \times c - d \div e\]
   \[3 \ 1 \ 4 \ 2\]

3) \[a \div (b + c) - d \% e\]
   \[2 \ 1 \ 4\]
In-Class Exercise

Determine the order of evaluation in the following expressions.

1) \( a + b + c + d + e \)
   1 2 3 4

2) \( a + b \times c - d / e \)
   3 1 4 2

3) \( a / (b + c) - d \% e \)
   2 1 4 3
Determine the order of evaluation in the following expressions.

1) \(a + b + c + d + e\)
   
   1 2 3 4

2) \(a + b \times c - d \div e\)
   
   3 1 4 2

3) \(a \div (b + c) - d \mod e\)
   
   2 1 4 3

4) \(a \div (b \times (c + (d - e)))\)
In-Class Exercise

- Determine the order of evaluation in the following expressions.

1) \( a + b + c + d + e \)  
   1 2 3 4

2) \( a + b * c - d / e \)  
   3 1 4 2

3) \( a / (b + c) - d \% e \)  
   2 1 4 3

4) \( a / (b * (c + (d - e))) \)  
   4 3 2 1
Expression Trees

- Evaluation order of an expression can also be shown using an expression tree.
- The operators *lower* in the tree have *higher* precedence for that expression.

```
(a + (b - c) / d) +
```

```
+   
  /  
-   d
  
   
  b  c
```
The assignment operator has the lowest operator precedence.

The entire right-hand side expression is evaluated first, then the result is stored in the original variable.

It is common for the right hand side and left hand sides of an assignment statement to contain the same variable.

```c
  count = count + 1;
```
Increment and Decrement Operators

- The **increment operator** (`++`) adds one to its operand.
  - The following statements produce the same result.
    ```java
    count++;  
count = count + 1;
    ```

- The **decrement operator** (`--`) subtracts one from its operand.
  - The following statements produce the same result.
    ```java
    count--;  
count = count - 1;
    ```
The increment and decrement operators can be applied in 
**postfix form**

```cpp
    count++;  count--;  // postfix form
```

or **prefix form**

```cpp
    ++count;  --count;  // prefix form
```

When used as part of a larger expression, the two can have 
different effects.

*Use with care!!*
Assignment Operators

- Java provides assignment operators to simplify expressions where we perform an operation on an expression then store the result back into that variable.
- Consider the following expression.
  ```java
  num = num + count;
  ```
- We can simplify this using the addition assignment operator.
  ```java
  num += count;
  ```
- Java provides the following assignment operators.
  - `+=` (string concat or addition), `-=`, `*=`, `/=`, `%=`
Data Conversion

- Sometimes we need to convert from one data type to another (e.g. float to int).
- These conversions do not change the type of a variable, they just convert it temporarily as part of a computation.
- **Widening conversions.** Safest. Go from small data type to large one.
  - e.g. short to int, int to double
- **Narrowing conversions.** Not so safe. Go from large data type to smaller one. Must be used carefully.
  - e.g. int to short, double to int
Data Conversions

- Assignment conversion.
- Promotion.
- Casting.
Assignment Conversion

- Assignment conversion occurs when one type is assigned to a variable of another.
- Only *widening conversions* can happen via assignment.
- For example:
  ```java
  float totalCost;
  int dollars;
  totalCost = dollars;
  ```
- The *value* stored in `dollars` is converted to a `float` before it is assigned to the `totalCost` variable.
- The `dollars` variable and the value stored in it are still `int` after the assignment.
Promotion

- Promotion happens automatically when operators in expressions convert their operands.

- For example:
  ```
  float sum;
  int count;
  double result = sum / count;
  ```

- The value of count is converted to a float before the division occurs.

- Note that a widening conversion also occurs when the result is assigned to result.
Casting

- **Casting** is the most powerful and potentially dangerous conversion technique.
- Explicitly perform *narrowing* and *widening* conversions.
- Recall our example from earlier:
  ```java
  int length = 15, sections = 2;
  float newLength = length / sections;
  ```
  Recall:
  If both operands of the division operator are integers, then the result will be an integer. If either or both operands used by an arithmetic operator are floating point, then the result will be a floating point.
  By casting one of the operands (length in this case), we get the desired result:
  ```java
  float newLength = ((float) length) / sections;
  ```
Casting

- **Casting** is the most powerful and potentially dangerous conversion technique.
- Explicitly perform *narrowing* and *widening* conversions.
- Recall our example from earlier:
  ```java
  int length = 15, sections = 2;
  float newLength = length / sections;
  ```
- Recall: *If both operands of the division operator are integers, then the result will be an integer. If either or both operands used by an arithmetic operator are floating point, then the result will be a floating point.*
Casting

- Casting is the most powerful and potentially dangerous conversion technique.
- Explicitly perform *narrowing* and *widening* conversions.
- Recall our example from earlier:
  ```java
  int length = 15, sections = 2;
  float newLength = length / sections;
  ```
- Recall: *If both operands of the division operator are integers, then the result will be an integer. If either or both operands used by an arithmetic operator are floating point, then the result will be a floating point.*
- By casting one of the operands (*length* in this case), we get the desired result
  ```java
  float newLength = ((float) length) / sections;
  ```
In-Class Exercise

Will the following program produce an accurate conversion (why or why not)?

```java
/**
 * Computes the Fahrenheit equivalent of a specific Celsius value using
 * the formula:
 * F = (9/5) * C + 32.
 */
public class TempConverter {
    public static void main(String[] args) {
        final int BASE = 32;
        double fahrenheitTemp;
        int celsiusTemp = 24; // value to convert
        fahrenheitTemp = celsiusTemp * 9 / 5 + BASE;
        System.out.println("Celsius Temperature: " + celsiusTemp);
        System.out.println("Fahrenheit Equivalent: " + fahrenheitTemp);
    }
}
```

1. Yes.
2. Sometimes.
3. Nope.
4. I have no idea.
Typically, we want our programs to interact with our users.

The Scanner class is part of the java.util class library. It must be imported.

It provides methods for reading input values of various types.

A Scanner object can read input from various sources (e.g. keyboard, file)

See Java 8 API docs: http://docs.oracle.com/javase/8/docs/api/java/util/Scanner.html
Create a new Scanner object that reads from the keyboard.

```java
Scanner scan = new Scanner(System.in);
```

The `new` operator creates a new Scanner object.

The `System.in` object represents keyboard input.

After the object is created, we can invoke various input methods it provides.
Example

- **Example**: Convert TempConverter.java to interactive program.
- **Example**: Echo.java
By default, **white space** is used to separate input elements (called **tokens**).

- White space includes:
  - Space characters (` ' `)
  - Tab characters (`\t`)
  - New line characters (`\n` and `\r`)

- The **next**, **nextInt**, **nextDouble**, etc. methods of the **Scanner** class read and return the next input tokens.

- See **Scanner** documentation for more details.
Example

Example: GasMileage.java
Exercises

- Read Chapter 3 of textbook.