Programming: Data and Interaction
UWL as Data

Calculating a student's age: Write out instructions to calculate a student’s age, given their birthday (i.e., year, month, day) and a value for today’s date. Avoid using words like “before” or “after”; instead, use words for numerical comparison (e.g., “greater than”, “less than or equal to”). Test your instructions with the following possibilities for today’s date:

- March 26, 2016
- January 26, 2016
- February 22, 2016
- February 24, 2016
- February 23, 2016

- birthday: Feb. 23, 1994
  - year: 1994
  - month: 2
  - day: 23
UWL as Data

- birthday: Feb. 23, 1994
  - year: 1994
  - month: 2
  - day: 23

1. Subtract the birthday year from today’s year.
2. a. If the birthday month is greater than today’s month, then subtract one from the result of step 1 to obtain the final answer.
   b. If the birthday month is the same as today’s month, and the birthday day is greater than today’s day, then subtract one from the result of step 1 to obtain the final answer.
   c. If you do not perform steps 2.a. or 2.b., then the result of step 1 is the final answer.
UWL as Data

1. Subtract the birthday year from today’s year.
2. a. If the birthday month is greater than today’s month, then subtract one from the result of step 1 to obtain the final answer.
   b. If the birthday month is the same as today’s month, and the birthday day is greater than today’s day, then subtract one from the result of step 1 to obtain the final answer.
   c. If you do not perform steps 2.a. or 2.b., then the result of step 1 is the final answer.

birthday:
  year: 1994  month: 2  day: 23

today’s date:
  year: 2016  month: 3  day: 26
1. Subtract the birthday year from today’s year.

2. a. If the birthday month is greater than today’s month, then subtract one from the result of step 1 to obtain the final answer.

   b. If the birthday month is the same as today’s month, and the birthday day is greater than today’s day, then subtract one from the result of step 1 to obtain the final answer.

   c. If you do not perform steps 2.a. or 2.b., then the result of step 1 is the final answer.

birthday:
   year: 1994  month: 2  day: 23

today’s date:
   year: 2016  month: 3  day: 26

1. 2016 - 1994 = 22
1. Subtract the birthday year from today’s year.

2. a. If the birthday month is greater than today’s month, then subtract one from the result of step 1 to obtain the final answer.

b. If the birthday month is the same as today’s month, and the birthday day is greater than today’s day, then subtract one from the result of step 1 to obtain the final answer.

c. If you do not perform steps 2.a. or 2.b., then the result of step 1 is the final answer.

birthday:
  year: 1994  month: 2  day: 23

today’s date:
  year: 2016  month: 3  day: 26

1. 2016 - 1994 = 22

2. a. 2 > 3?  no
1. Subtract the birthday year from today’s year.
2. a. If the birthday month is greater than today’s month, then subtract one from the result of step 1 to obtain the final answer.
   b. If the birthday month is the same as today’s month, and the birthday day is greater than today’s day, then subtract one from the result of step 1 to obtain the final answer.
   c. If you do not perform steps 2.a. or 2.b., then the result of step 1 is the final answer.

---

**birthday:**
- year: 1994
- month: 2
- day: 23

**today’s date:**
- year: 2016
- month: 3
- day: 26

---

1. 2016 - 1994 = 22
2. a. 2 > 3?  no
   b. 2 = 3 and 23 > 26?  no
UWL as Data

1. Subtract the birthday year from today’s year.
2. a. If the birthday month is greater than today’s month, then subtract one from the result of step 1 to obtain the final answer.
   b. If the birthday month is the same as today’s month, and the birthday day is greater than today’s day, then subtract one from the result of step 1 to obtain the final answer.
   c. If you do not perform steps 2.a. or 2.b., then the result of step 1 is the final answer.

birthday:
  year: 1994  month: 2  day: 23

today’s date:
  year: 2016  month: 3  day: 26

1. 2016 - 1994 = 22
2. a. 2 > 3?  no
   b. 2 = 3 and 23 > 26?  no
   c. neither steps 2.a. or 2.b. performed?  yes
UWL as Data

1. Subtract the birthday year from today’s year.
2. a. If the birthday month is greater than today’s month, then subtract one from the result of step 1 to obtain the final answer.
   b. If the birthday month is the same as today’s month, and the birthday day is greater than today’s day, then subtract one from the result of step 1 to obtain the final answer.
   c. If you do not perform steps 2.a. or 2.b., then the result of step 1 is the final answer.

birthday:
   year: 1994   month: 2   day: 23

today’s date:
   year: 2016   month: 3   day: 26

1. 2016 - 1994 = 22
2. a. 2 > 3?   no
   b. 2 = 3 and 23 > 26?   no
   c. neither steps 2.a. or 2.b. performed?   yes

answer = 22
1. Subtract the birthday year from today’s year.
2. a. If the birthday month is greater than today’s month, then subtract one from the result of step 1 to obtain the final answer.
b. If the birthday month is the same as today’s month, and the birthday day is greater than today’s day, then subtract one from the result of step 1 to obtain the final answer.
c. If you do not perform steps 2.a. or 2.b., then the result of step 1 is the final answer.

birthday:
  year: 1994  month: 2  day: 23
today’s date:
  year: 2016  month: 2  day: 22
UWL as Data

1. Subtract the birthday year from today’s year.
2. a. If the birthday month is greater than today’s month, then subtract one from the result of step 1 to obtain the final answer.
   b. If the birthday month is the same as today’s month, and the birthday day is greater than today’s day, then subtract one from the result of step 1 to obtain the final answer.
   c. If you do not perform steps 2.a. or 2.b., then the result of step 1 is the final answer.

<table>
<thead>
<tr>
<th>birthday:</th>
<th>today’s date:</th>
</tr>
</thead>
<tbody>
<tr>
<td>year: 1994  month: 2  day: 23</td>
<td>year: 2016  month: 2  day: 22</td>
</tr>
</tbody>
</table>

1. 2016 - 1994 = 22
UWL as Data

1. Subtract the birthday year from today’s year.
2. a. If the birthday month is greater than today’s month, then subtract one from the result of step 1 to obtain the final answer.
   b. If the birthday month is the same as today’s month, and the birthday day is greater than today’s day, then subtract one from the result of step 1 to obtain the final answer.
   c. If you do not perform steps 2.a. or 2.b., then the result of step 1 is the final answer.

birthday:
year: 1994  month: 2  day: 23

today’s date:
year: 2016  month: 2  day: 22

1. 2016 - 1994 = 22
2. a. 2 > 2?  no
UWL as Data

1. Subtract the birthday year from today’s year.
2. a. If the birthday month is greater than today’s month, then subtract one from the result of step 1 to obtain the final answer.
   b. If the birthday month is the same as today’s month, and the birthday day is greater than today’s day, then subtract one from the result of step 1 to obtain the final answer.
   c. If you do not perform steps 2.a. or 2.b., then the result of step 1 is the final answer.

birthday:  
  year: 1994  month: 2  day: 23

today’s date:  
  year: 2016  month: 2  day: 22

1. 2016 - 1994 = 22
2. a. 2 > 2?  no
   b. 2 = 2 and 23 > 22?  yes
UWL as Data

1. Subtract the birthday year from today’s year.
2. a. If the birthday month is greater than today’s month, then subtract one from the result of step 1 to obtain the final answer.
   b. If the birthday month is the same as today’s month, and the birthday day is greater than today’s day, then subtract one from the result of step 1 to obtain the final answer.
   c. If you do not perform steps 2.a. or 2.b., then the result of step 1 is the final answer.

birthday:
   
   today’s date:

1. 2016 - 1994 = 22
2. a. 2 > 2? no
   b. 2 = 2 and 23 > 22? yes
   answer = 22 - 1 = 21
UWL as Data

1. Subtract the birthday year from today’s year.
2. a. If the birthday month is greater than today’s month, then subtract one from the result of step 1 to obtain the final answer.
   b. If the birthday month is the same as today’s month, and the birthday day is greater than today’s day, then subtract one from the result of step 1 to obtain the final answer.
   c. If you do not perform steps 2.a. or 2.b., then the result of step 1 is the final answer.

birthday:
  year: 1994  month: 2  day: 23

today's date:
  year: 2016  month: 2  day: 22

1. 2016 - 1994 = 22
2. a. 2 > 2?  no
   b. 2 = 2 and 23 > 22?  yes
      answer = 22 - 1 = 21
   c. neither steps 2.a. or 2.b. performed?  no
1. Subtract the birthday year from today’s year.

2. a. If the birthday month is greater than today’s month, then subtract one from the result of step 1 to obtain the final answer.

b. If the birthday month is the same as today’s month, and the birthday day is greater than today’s day, then subtract one from the result of step 1 to obtain the final answer.

c. If you do not perform steps 2.a. or 2.b., then the result of step 1 is the final answer.

- when the birthday month has not yet occurred
- when the birthday month is today’s month, but the birthday day has not yet occurred
- when the birthday has already passed
# UWL as Object-Oriented Data

<table>
<thead>
<tr>
<th>Objects</th>
<th>Attributes</th>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>professors</td>
<td>- first name</td>
<td>- display schedule of classes</td>
</tr>
<tr>
<td></td>
<td>- last name</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- department</td>
<td>- calculate age</td>
</tr>
<tr>
<td></td>
<td>- list of <strong>classes</strong> this semester</td>
<td>- display schedule of classes</td>
</tr>
<tr>
<td>students</td>
<td>- first name</td>
<td>- calculate classes left</td>
</tr>
<tr>
<td></td>
<td>- last name</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- major</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- list of <strong>classes</strong> this semester</td>
<td></td>
</tr>
<tr>
<td>classes</td>
<td>- department (e.g., CS)</td>
<td>- calculate number of seats left</td>
</tr>
<tr>
<td></td>
<td>- number (e.g., 120)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- section (e.g., 1)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- <strong>professor</strong> of record</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- list of <strong>students</strong> enrolled</td>
<td></td>
</tr>
</tbody>
</table>
UWL as Object-Oriented Data

**Objects**

- professors
- students
- classes

**objects/classes:** allows us to organize data and actions to be performed on that data based on real-world phenomena

Comprised of two parts:

1. **attributes/data members:** data that describes the object
2. **methods/functions:** instructions for calculations that can be performed on the object’s attributes
UWL as Object-Oriented Data

**Objects**

- **professors**
  - first name
  - last name
  - department
  - list of classes this semester

- **students**
  - first name
  - last name
  - major
  - list of classes this semester

- **classes**
  - department (e.g., CS)
  - number (e.g., 120)
  - section (e.g., 1)
  - professor of record
  - list of students enrolled

**Attributes**

**Methods**

- display schedule of classes
- calculate age
- display schedule of classes
- calculate classes left
- calculate number of seats left
- order students by grade
Methods

Methods are a **named set of instructions**

Method: calculating a person’s age (given their birthday and today’s date)

  instruction 1: subtract the person’s birth year from the current year

  instruction 2: determine which part of instruction 2 (a, b, or c) to execute and perform it
Statements

`statement`: the unit of instruction in programming

enables us to give commands to the computer

Crux of all programming languages

Programming is about the use of statements to solve problems

In Java, statements *always* end with a semicolon

<instruction 1>;
<instruction 2>;
<instruction 3>;}
Program Structure

```java
/**
 * Our first program
 */

public class ExampleClass {

    public static void main(String[] args) {
        // Your code goes here!
    }
}
```
Program Structure: Class

```java
/**
 * Our first program
 */
public class ExampleClass {
    public static void
    // Your code goes here!
}
```

Provides a name for the program
One program per class
For now, always created with
public class <className>
replace <className> with the
program name
<className> must match the name of
the file!
Program Structure: `main` Method

```java
/**
 * Our first program
 */
public class ExampleClass {

    public static void main(String[] args) {
        // Your code goes here!
    }
}
```

Denotes where the program will start executing

Only one `main` method per program

Always created with

```java
public static void main(String[] args)
```
Program Structure: Comments

Allows us to annotate our program not interpreted as code/instructions completely ignored by the computer

Comments are often inserted on their own line(s)
Definition: Comments

inline comment

```
// Begins with two slashes; this comment lasts until the end of the line
```

block comment

```
/**
 * This is a block comment.
 * Typically used at the top of a class file or before methods,
 * and can span multiple lines.
 * Starts with a single slash followed by an asterisk,
 * and ends with an asterisk followed by a slash.
 */
```
Program Structure: Code Blocks

**/  
* Our first program  
*/

public class ExampleClass {

    public static void main(String[] args) {
        // Your code goes here!
    }
}

Defined by matching opening and closing curly bracket (e.g., { & })

Can be nested

innermost opening curly bracket matches innermost closing curly bracket
on to

**data** and **interaction**
How can I take the **data I have** and transform it into the **data I need**?
<table>
<thead>
<tr>
<th>Text</th>
<th>Numbers</th>
<th>Logical Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Carpe Diem”</td>
<td>42</td>
<td>true</td>
</tr>
<tr>
<td></td>
<td>3.14159</td>
<td>false</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>text</td>
<td>numbers</td>
<td>logical values</td>
</tr>
</tbody>
</table>
Data

“Carpe Diem”  42  3.14159  true

text  numbers  logical values
Textual Data

Good for data not easily represented by numbers

e.g., names, majors, descriptions

*string literal*: a sequence of characters that should be interpreted as data, not instructions

colloquially, we call these *strings*
Strings

"This is a string."

Quotes define the beginning and end of a string

are not part of the string itself

Can include any standard characters

e.g., numbers, spaces, punctuation

Called a *string literal* since the data is exactly what is stored between quotes
Console

Allows us to communicate textually with a Java program

Java produces output with System.out (sometimes referred to as standard output)

Java reads in input with System.in (sometimes referred to as standard input)

```java
public class ExampleClass {
    public static void main(String[] args) {
        // Your code goes here!
    }
}
```
Definition: String Output

print statement: prints `<string>` to the console

```
System.out.print(<string>);
```

println statement: prints `<string>` to the console, then moves to the next line

```
System.out.println(<string>);
```

Nota Bene (N.B.): anything with angle brackets should be replaced by something

N.B.: the rest of the statement needs to be exactly as shown here: capitalization, spelling
Printing Strings

```java
public class Name {
    public static void main(String[] args) {
        System.out.println("Allie Sauppe");
    }
}
```

Allie Sauppe
Printing Strings

```java
public class Name {
    public static void main(String[] args) {
        System.out.print("Allie Sauppe, CS");
    }
}
```

Allie Sauppe, CS
public class Name {
    public static void main(String[] args) {
        System.out.println("Allie Sauppe");
    }
}

Allie Sauppe
^
Sequential Execution

Instructions start executing in `main` method

Execute one at a time, in order, starting at top of `main`

Order matters!

- changing the order of instructions will often change the functionality of the program
- particularly important when printing to console — cannot go backwards
public class Name {
    public static void main(String[] args) {
        System.out.print("Allie Sauppe");
        System.out.print(", CS");
    }
}
public class Name {
    public static void main(String[] args) {

        > System.out.print("", CS");
        > System.out.print("Allie Sauppe");
        >

    }
}

, CSAllie Sauppe,
public class Name {
    public static void main(String[] args) {
        System.out.print("Allie Sauppe");
        System.out.println(", CS");
        System.out.print("UW-La Crosse");
    }
}
Exercise: Adding Quotation Marks

Use `print` and `println` statements to display the following:

"I'll be back."
- The Terminator

```java
public class Name {
    public static void main(String[] args) {
        System.out.println(""I'll be back."");
        System.out.print("- The Terminator");
    }
}
```

this will not work!
Escape Character

Allows us to *escape* the string with a backslash (the *escape character*)

Escape character + next character are interpreted together, non-literally to form an *escape sequence*

Common escape sequences:

\" //prints a double quotation mark
\' //prints a single quotation mark
\n //prints a newline
\t //prints a tab
Escape Character

Allows us to escape the string with a backslash (the escape character).

Escape character + next character are interpreted together, non-literally, form an escape sequence.

Common escape sequences:

  \\  //prints a double quotation mark
  \'  //prints a single quotation mark
  \n  //prints a newline
  \t  //prints a tab
  \\  //prints a backslash
Example: Using Escape Sequences

Use `print` and `println` statements to display the following:

"I'll be back."
- The Terminator

```java
public class Name {
    public static void main(String[] args) {
        System.out.println("\"I'll be back.\"");
        System.out.print("- The Terminator");
    }
}
```
Variables

`variable`: a piece of computer memory that holds data

Two parts to every variable:

1. *identifier*: the name by which we refer to the variable
2. *data type*: the type of data the variable holds (e.g., string, number, boolean)
Identifiers

identifier: name we use to refer to parts of code
e.g., variables, classes, methods

Must follow a few rules:

start with an alphabetic character (a-z, A-Z), underscore (_), or dollar sign ($)
contain only alphanumerical characters (a-z, A-Z, 0-9), underscore (_), or dollar sign ($)

Should be descriptive

No spaces!

use camelcase to name variables
Camelcase

Might want to give identifiers containing multiple words

mybirthday
yourbirthday

camelcase: only first letter of each word is uppercase

MyBirthday //capitalize first letter for classes
myBirthday //lowercase first letter for variables, methods
Identifiers

Case matters

mybirthday, myBirthday, MyBirthday and MYBIRTHDAY are all unique variable names

Identifiers cannot be reserved keywords

public int
protected double
private boolean
static new
void return
final ...
**Data Type**

data type: the type of data the variable holds; defines what actions can be performed on it

  e.g., we can divide one number by another, we can’t divide one string by another

Cannot be changed once variable is created
# Types of Data Type

Two categories: *primitive type* and *class type*

<table>
<thead>
<tr>
<th>Primitives</th>
<th>Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>represents basic data types</td>
<td>represents more complex data</td>
</tr>
<tr>
<td>examples:</td>
<td>examples:</td>
</tr>
<tr>
<td>char  //holds a single character</td>
<td>String  //** holds textual data</td>
</tr>
<tr>
<td>int    //holds integer values</td>
<td>Scanner //reads input</td>
</tr>
<tr>
<td>double //holds decimal values</td>
<td>Date //represents day/month/year</td>
</tr>
<tr>
<td>boolean //holds true/false values</td>
<td>Math //complex mathematical ops</td>
</tr>
</tbody>
</table>
Using Variables

Two parts to variable use:

1. **declaring the variable**: defines the variable's data type and identifier

2. **initializing the variable**: sets the variable to some value; sets it up to be used

Variables must be...

- declared before they can be initialized
- initialized before they can be used

Can be done separately or together

Declaration must happen exactly once for each variable
Definition: Variable Declaration

declare a single variable

```plaintext
<dataType> <identifier>;
```

declare multiple variables of the same type

```plaintext
<dataType> <identifier>, <identifier>, <identifier>;
```

N.B.: remember, anything in angle brackets should be completely replaced! (including the brackets)
Example: Variable Declaration

declare a single variable

```java
int age;
double height;
String name;
```

declare multiple variables of the same type

```java
int day, favoriteNumber;
double temp, weight;
String firstName, lastName, middleName;
```
Example: Variable Declaration

public class Person {
    public static void main(String[] args) {
        int age;
        double height;
        String firstName;
    }
}

memory

<table>
<thead>
<tr>
<th>age (int)</th>
</tr>
</thead>
<tbody>
<tr>
<td>height (double)</td>
</tr>
<tr>
<td>firstName (String)</td>
</tr>
</tbody>
</table>
Definition: Primitive Variable Initialization

initialize a primitive variable

<identifier> = <value>;

N.B.: the data type associated with the identifier **must** match the data type of the value
Example: Primitive Variable Initialization

initialize a primitive variable

```java
firstName = "James";
```

this works because we are initializing a String variable with a String value
public class Person {
    public static void main(String[] args) {
        String firstName, lastName;
        int age;
        firstName = "James";
        age = 42;
    }
}
Definition: Combining Declaration & Initialization

declare & initialize a single primitive variable

<dataType> <identifier> = <value>;

declare & initialize multiple primitive variables of the same type

<dataType> <identifier> = <value>, <identifier> = <value>, <identifier> = <value>,
Example: Combining Declaration & Initialization

declare & initialize a single primitive variable

```java
String firstName = "James";
```

declare & initialize multiple primitive variables of the same type

```java
String firstName = "James", lastName = "Kirk", middleName;
```
Example: Combining Declaration & Initialization

```java
class Person {
    public static void main(String[] args) {
        String firstName = "James", middleName, lastName = "Kirk";
        middleName = "Tiberius";
    }
}
```
Definition: String Output

print statement: prints $<$String$>$ to the console

```
System.out.println($<$String$>$);
```

println statement: prints $<$String$>$ to the console, then moves to the next line

```
System.out.println($<$String$>$);
```
public class Person {
    public static void main(String[] args) {

        String firstName = "James", lastName = "Kirk";
        int age = 42;

        System.out.println(firstName);
        System.out.println("James");
        System.out.println(lastName);
        System.out.println("Kirk");
        System.out.println(age);
        System.out.println("42");
    }
}
Definition: Primitive Variable Assignment

assign a new value to a variable

```
<identifier> = <value>;
```

N.B.: the data type associated with the identifier must match the data type of the value

Variable initialization versus assignment

- *initialization* is the first time a value is assigned to a variable
- *assignment* is overwriting the current value with a new value

In practice, look the same
Primitive Variable Assignment

```java
public class Person {
    public static void main(String[] args) {
        String firstName = "James", lastName = "Kirk", middleName;
        System.out.println(firstName);
        System.out.println(lastName);
        firstName = "Jim";
        System.out.println(firstName);
    }
}
```

<table>
<thead>
<tr>
<th>memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>firstName (String)</td>
</tr>
<tr>
<td>lastName (String)</td>
</tr>
<tr>
<td>middleName (String)</td>
</tr>
</tbody>
</table>

James
Kirk
Jim
Primitive Variable Assignment

public class Person {
    public static void main(String[] args) {
        String firstName = "James", lastName = "Kirk", middleName;
        System.out.println(firstName);
        System.out.println(lastName);
        firstName = lastName;
        System.out.println(firstName);
    }
}

memory

firstName (String)
"James"

lastName (String)
"Kirk"

middleName (String)

James
Kirk
Kirk
String Methods

Text is one of our fundamental units of data

Several ways we might want to manipulate our text

Examples:

- change letters to all upper or lowercase
- isolate a small part of the text
- find a particular letter or number in a text
- replace some part of the text
Strings

"This is a string."

these are the index values for the String
Methods

Methods have four main characteristics we should know.

For any given method:

- what is it called?
- what does it do?
- what type of input does it need? (called parameters)
- what type does it give back? (i.e., what does it return?)
Definition: String Methods

+: concatenates two String values together

```
<String> + <String>;
```

length: returns the length of <String> (i.e., how many characters)

```
<String>.length();
```

substring: returns part of <String> from index <int1> to index <int2>

```
<String>.substring(<int1>, <int2>);
```
Concatenation (+)

*concatenate*: to join two Strings together into one String

*arguments*: the two Strings to join together

*returns*: a single String

```java
<String> + <String>
```

```java
String str1 = "Hello", str2 = "World";
String exampleConcat = str1 + str2;
System.out.print(exampleConcat);
```

HelloWorld
Definition: String Methods

+: concatenates two String values together

```
<String> + <String>;
```

length: returns the length of <String> (i.e., how many characters)

```
<String>.length();
```

substring: returns part of <String> from index <int1> to index <int2>

```
<String>.substring(<int1>, <int2>);
```
Concatenation (+)

**concatenate**: to join two Strings together into one String

**arguments**: the two Strings to join together

**returns**: a single String

```java
"String" + "String";

> String str1 = "Hello", str2 = "World";
> String exampleConcat = str1 + str2;
> System.out.print(exampleConcat);

HelloWorld
```
Concatenation (+)

**concatenate**: to join two Strings together into one String

**arguments**: the two Strings to join together

**returns**: a single String

```java
<String> + <String>;

String str1 = "Hello", str2 = "World";
String exampleConcat = str1 + " " + str2;
System.out.print(exampleConcat);
```

```
Hello World
```

length

arguments: none

returns: the length (int) of the String (i.e., the number of characters)

```
<String>.length();

String exampleStr = "Hello, world!";
int len = exampleStr.length();
System.out.print(len);
```

```
13
```

memory

```
len (int)
13
exampleStr (String)
"Hello, world!"
```
substring

arguments: the beginning index <int1> (inclusive), the ending index <int2> (exclusive)

returns: the String specified by the beginning and end index

```java
<String>.substring(<int1>, <int2>);
```

```java
> String exStr = "All the king's men.";
> String exSubStr = exStr.substring(4, 14);
> System.out.print(exSubStr);
> the king's
```

memory

```java
exStr (String)
"All the king's men."
```

```java
exSubStr (String)
"the king's"
```
**Definition: String Methods**

**indexOf**: returns the index (`<int>`) of the first occurrence of `<char>`

```java
/String/.indexOf(<char>);
```

**charAt**: returns the `<char>` present at index `<int>`

```java
/String/.charAt(<int>);
```

**replaceAll**: replace every occurrence of `<String1>` with `<String2>`

```java
/String/.replaceAll(<String1>, <String2>);
```
indexOf

**arguments**: the char to look for `<char>` (case sensitive!)

**returns**: the index `<int>` of the first occurrence of char

```java
例 String exampleStr = "Hello, home!";
int index = exampleStr.indexOf('h');
System.out.print(index);
```
indexOf

**arguments:** the char to look for `<char>` (case sensitive!)

**returns:** the index `<int>` of the first occurrence of char

```java
<String>.indexOf(<char>);

String exampleStr = "Hello, home!";
int index = exampleStr.indexOf('h');
> System.out.print(index);
> 7
```

memory

<table>
<thead>
<tr>
<th>exampleStr (String)</th>
<th>“Hello, home!”</th>
</tr>
</thead>
<tbody>
<tr>
<td>index (int)</td>
<td>7</td>
</tr>
</tbody>
</table>
indexOf

**arguments:** the char to look for `<char>` (case sensitive!)

**returns:** the index (`<int>`) of the first occurrence of char

```java
<String>.indexOf(<char>);
String exampleStr = "Hello, home!";
int index = exampleStr.indexOf('H');
System.out.print(index);
```

**memory**

- exampleStr (String)
  - "Hello, home!"
- index (int)
indexOf

arguments: the char to look for <char> (case sensitive!)
returns: the index (<int>) of the first occurrence of char

```java
<String>.indexOf(<char>);

String exampleStr = "Hello, home!";
int index = exampleStr.indexOf('H');
System.out.println(index);
```

memory

```
exampleStr (String)
"Hello, home!"

index (int)
0
```
charAt

arguments: a specific index in the String <int>
returns: the char at that index

```
<String>.charAt(<int>);
```

```
> String exampleStr = "Hello, home!";
> char charPos = exampleStr.charAt(5);
> System.out.print(charPos);
```
**charAt**

**arguments:** a specific index in the String `<int>`

**returns:** the char at that index

```java
<String>.charAt(<int>);
```

```java
String exampleStr = "Hello, home!";
char charPos = exampleStr.charAt(5);
System.out.print(charPos);
```
replaceAll

arguments: the String to replace is <String1>, the replacement String is <String2>
returns: a String with every occurrence of <String1> replaced by <String2>

```java
<String>.replaceAll(<String1>, <String2>);
```

```java
String exampleStr = "She sells seashells";
String newStr = exampleStr.replaceAll("ll", "_!!_");
System.out.print(newStr);
```

```
She se_!!_s seashe_!!_s
```
Order of Evaluation

In order to set/change the value of a variable, = must be used!

Java will evaluate right of equal sign first

- moves left to right
- evaluates inner parentheses before outer parentheses

```java
String exampleStr = "She sells seashells";
System.out.print(exampleStr.replaceAll("ll", "_!!_"));
```
In order to set/change the value of a variable, = must be used!

Java will evaluate right of equal sign first

moves left to right

evaluates inner parentheses before outer parentheses

```java
String exampleStr = "She sells seashells";
System.out.print(exampleStr.replaceAll("ll", "_!!_"));
```

N.B.: we know print methods must have some string argument
In order to set/change the value of a variable, = must be used!

Java will evaluate right of equal sign first

- moves left to right
- evaluates inner parentheses before outer parentheses

```
String exampleStr = "She sells seashells";
System.out.print(exampleStr.replaceAll("ll", "_!!_"));
```

this statement evaluates to a string, so we can use it here
In order to set/change the value of a variable, = must be used!

Java will evaluate right of equal sign first

String exampleStr = "She sells seashells";
String exampleStr2 = "and other things";

```java
exampleStr = exampleStr.replaceAll("ll", "_!!_") + exampleStr2;
```

memory

- exampleStr (String)
  - "She sells seashells"

- exampleStr2 (String)
  - "and other things"
Order of Evaluation

In order to set/change the value of a variable, = must be used!

Java will evaluate right of equal sign first

```java
String exampleStr = "She sells seashells";
String exampleStr2 = "and other things";
exampleStr = exampleStr.replaceAll("ll", "_!!_") + exampleStr2;
```

memory

```plaintext
exampleStr (String)
"She sells seashells"

exampleStr2 (String)
"and other things"
```
Order of Evaluation

In order to set/change the value of a variable, = must be used!

Java will evaluate right of equal sign first

```java
String exampleStr = "She sells seashells";
String exampleStr2 = "and other things";
exampleStr = exampleStr.replaceAll("ll", "_!!_") + exampleStr2;
```

memory

<table>
<thead>
<tr>
<th>exampleStr (String)</th>
<th>exampleStr2 (String)</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;She sells seashells&quot;</td>
<td>&quot;and other things&quot;</td>
</tr>
</tbody>
</table>
Order of Evaluation

In order to set/change the value of a variable, = must be used!

Java will evaluate right of equal sign first

```java
String exampleStr = "She sells seashells";
String exampleStr2 = "and other things";

exampleStr = exampleStr.replaceAll("ll", "_!!_") + exampleStr2;
```

memory

- `exampleStr` (String): "She sells seashells"
- `exampleStr2` (String): "and other things"
Order of Evaluation

In order to set/change the value of a variable, = must be used!

Java will evaluate right of equal sign first

```java
String exampleStr = "She sells seashells";
String exampleStr2 = "and other things";

exampleStr = exampleStr.replaceAll("ll", "_!!_") + exampleStr2;
```

memory

```
exampleStr (String)
"She sells seashells"

exampleStr2 (String)
"and other things"
```
Order of Evaluation

In order to set/change the value of a variable, = must be used!
Java will evaluate right of equal sign first

```java
String exampleStr = "She sells seashells";
String exampleStr2 = "and other things";
exampleStr = exampleStr.replaceAll("ll", "_!!_") + exampleStr2;
exampleStr
```

memory

```
exampleStr (String)
"She sells seashells"
```

```
exampleStr2 (String)
"and other things"
```
Order of Evaluation

In order to set/change the value of a variable, = must be used!

Java will evaluate right of equal sign first

```java
String exampleStr = "She sells seashells";
String exampleStr2 = "and other things";

exampleStr = exampleStr.replaceAll("ll", "_!!_") + exampleStr2;
"She se_!!_s seashe_!!_s"_!!_sand other things" + "and other things"
```
Putting It All Together
The Scanner Class

Multiple ways to read input from a user

In this course, we’ll use the Java-provided `Scanner` class

our first class data type!

Provides input from the console
Using the Scanner Class

```java
import java.util.Scanner;

public class Person {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        String firstName;

        System.out.print("What is your first name? ");
        firstName = scan.nextLine();
        System.out.print("Your name is ");
        System.out.print(firstName);
    }
}
```

What is your first name? Jim
Your name is Jim
import Statements

import java.util.Scanner;

public class
    public static void
        Scanner
            String
                System
                    firstName
                        System
                            System

} 

What is your first name? Jim
Your name is Jim

Enables your program to leverage additional functionality either from within Java, or from a third-party source

Eclipse will help you find what imports you need
Definition: Variable Declaration

declare a single variable

```
<dataType>  <identifier>;  
```

declare multiple variables of the same type

```
<dataType>  <identifier>, <identifier>, <identifier>;  
```
**Definition: Object Variable Instantiation**

instantiate an object variable

\[
<\text{identifier}> = \text{new } <\text{dataType}>(<\text{arguments}>);
\]

N.B.: the data type associated with the identifier **must** match this data type

N.B.: arguments provide details necessary to create/use the object; will be specific to each type of object

We initialize primitive variables

We instantiate object variables

Same basic idea — setting the variable up for use
Definition: Combining Declaration & Instantiation

declare & instantiate a single object variable

```javascript
<dataType> <identifier> = new <dataType>(<arguments>);
```

declare & instantiate multiple object variables of the same type

```javascript
<dataType> <identifier> = new <dataType>(<arguments>), <identifier>;
```
Definition: Scanner Creation

declare & instantiate a single object variable

```java
<dataType> <identifier> = new <dataType>(<arguments>);
```

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

N.B.: this works because the data type associated with the identifier matches this data type

N.B.: for Scanner objects, we need to define where we are receiving input from; `System.in` specifies the console
Definition: Calling an Object’s Methods

calls \texttt{<methodName>}, specifying \texttt{<arguments>} if necessary

\texttt{<identifier>.<methodName>(<arguments>)};

dot notation says “we want to perform the set of instructions associated with \texttt{<methodName>}, and that this method is available for \texttt{<identifier>}'s data type”

we refer to this process as \textit{calling a method}
Definition: Scanner Methods

nextLine: reads in a String until a linebreak

```java
scan.nextLine();
```

nextInt: reads in a single int until whitespace (i.e., one number)

```java
scan.nextInt();
```

next: reads in a String until whitespace (i.e., one word)

```java
scan.next();
```
Definition: Method Returns

Once a method finishes it’s calculation, it will \textit{return} the result of the calculation to your program.

- the value returned will have a specific data type.
- not all methods will return a value.

```java
scanner.nextLine();  //returns a String
scanner.nextInt();   //returns an int
scanner.next();      //returns a String
```
Using the Scanner Class

```java
import java.util.Scanner;

public class Person {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        String firstName;

        System.out.print("What is your first name? ");
        firstName = scan.nextLine();
        System.out.print("Your name is ");
        System.out.print(firstName);
    }
}
```

What is your first name? Jim
Your name is Jim