THE UNIVERSITY OF ALABAMA AT BIRMINGHAM.

Welcome to CS203
Object-Oriented Programming

Lab 1

Link to Syllabus

The TAs hold Office Hours

- Jesse (jesselam@uab.edu)
- Brandon (vpham@uab.edu)
- Noah (ndia@uab.edu)
- Emanuel (emanuels@uab.edu)

Our Tutor holds SI Sessions

- Debbie (debbiefu@uab.edu)

How to be successful in CS203 and have the most fun

- Start homework as soon as possible
    - These assignments will be much larger than CS103
- Begin with design (this is hard since we all want to start with code)
    - Divide the problem into small pieces, solve, integrate as you go
    - Use pen and paper and visually represent your logic
- Learn how to debug and test (learn this early to make life easier)
- Join us during Office Hours and Tutoring Sessions!
- Google, google, google, **it's okay to use Google** (just cite your sources)
- **You should be programming outside of this class** (ask us for ideas!)

When Asking Coding Questions

- Before you even ask questions, **do your own research and Google**
  - Please don't ask for answers that can be found on Google
- **Introduce the problem** before you include the code
  - Describe your problem and introduce some background context
  - Explain how you encountered the problem and any difficulties that prevent you from solving it yourself
  - Describe what you have done so far to try to fix the problem
- **Try not to just copy in your entire program**!
  - Include enough code to allow us to reproduce the problem

THE UNIVERSITY OF ALABAMA AT BIRMINGHAM

Environment Set-Up

Canvas Course > Files > lab > lab01 > lab1.pdf

How to create, compile, and run a Java program

Code

1 Open text editor of choice, and write your Java program

```
public class ClassName{
  public static void main(String[] args){
    System.out.println("Hello World!");
  }
}
```

1b Save your file as ***filename*.java**
Note: *filename* needs to be the same as your class name

Compile

2 Open your terminal, (Command Prompt)

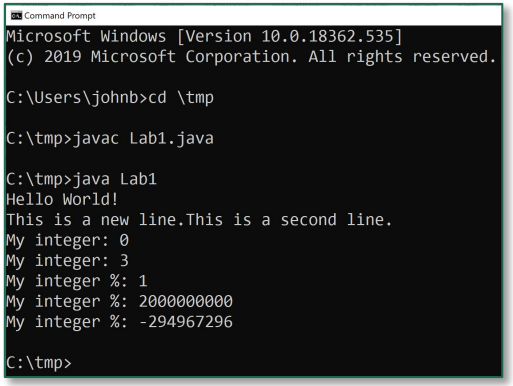
2b Navigate to the folder where your *filename*.java is stored

2c Compile via command:
javac *filename*.java
(this will create a .class file)

2d If there are errors, correct filename.java, Save, and re-compile

Run

3 In the same terminal window,
In the same folder,
Run via command:
java *filename*



In Eclipse, hit the "little green button", it will save, compile, (and if no errors), run. Errors or output appear in the Console

Typical Workflow – how to create and run your code

1. For each HW/Lab, create a new Project in the Package Explorer on the left
   - Name the Project, e.g.: Lab01 or HW1
   - Capitalize Project names!
2. Create a new Package inside the Project “src” folder
   - Name the Package something related to the assignment, e.g.: Lab01 or HW1
   - Capitalize Package names!
3. Inside the package, create a new Class (.java file)
   - Write your code inside this file
   - Capitalize Class names!
   - Create as many Class files as you need for your assignment (some may only need just one!)
4. Click on the ‘green-play-button’ to Save/Compile/Run

Submitting your code

When your code has been thoroughly tested & working correctly, then submit:

- Right-click on the Package
- Select “Export”
- Select “Archive file”
- Click “Next”
- (Change name of zip file to match submission instructions, and change location, if desired)
- Click “Finish”
- Submit .zip file to Canvas
- No penalty for multiple submissions, we’ll grade the last submission received before deadline

THE UNIVERSITY OF ALABAMA AT BIRMINGHAM.

Java Syntax

Example: Greeter.java

- This is a java program called Greeter.java which will print “Hello World!” into the console when you run it

```
public class Greeter {
      public static void main(String[] args) {
            System.out.println(“Hello World!”);
      }
}
```

Class

Warning: do not name your class “String”

- All Java code must be contained inside a Class
- The Class name should always start with an uppercase letter
- The name of the java file must be the same as the class name!
  - Otherwise, your code will not work!
  - If the class is called Greeter, save the file as Greeter.java

```
public class Greeter {
    public static void main(String[] args) {
        System.out.println(“Hello World!”);
    }
}
```

main method

- The main method is the starting point of your Java program
- The main method is required for any Java program
  - You need at least one main method to run the program!

```
public class Greeter {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

System.out.println(“Hello World!”)

- The println() method is used to print a String into the console

```
public class Greeter {
    public static void main(String[] args) {
        System.out.println(“Hello World!”);
    }
}
```

White space – semicolon ;

- Semi colons mark the end of a code statement
  - Remember, each code statement must end with a semicolon!

```
public class Greeter {
      public static void main(String[] args) {
            System.out.println("Hello World!");
      }
}
```

White space - curly braces { }

- White space does NOT matter in Java! White space did matter in Python.
  - White space was important to show where blocks of code were in Python
  - A block in coding is a grouping of two or more statements
- Curly braces mark the beginning and the end of a block of code
  - Any thing inside the curly braces circled pink is contained within the Greeter class
  - Any thing inside the curly braces circled blue is contained within the main method

```
public class Greeter {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

etc.

- There are other strange keywords before the Class name and the main method like “public”, “static”, or “void”
  - You don’t have to know how they work for now
  - You will learn more and more about them as you progress through this class!

```
public class Greeter {
        public static void main(String[] args) {
                System.out.println(“Hello World!”);
        }
}
```

THE UNIVERSITY OF ALABAMA AT BIRMINGHAM.

Java Programming

Comments

- Use comments to:
  - Explain code
  - Prevent execution of code
- Single-line comments
  - Two forward slashes
- Multi-line comments
  - Start with /*
  - End with */
- Use “CTRL + /” to comment out multiple lines!

```
// this is a single-line comment

/*
    this
    is a
    multi-line
    comment
*/
```

Variables

```
type variableName = value;
```

```
int num = 1;
boolean flag = true;
String name = “Jesse”;
```

- Value - a piece of information or data
  - 353, 9.3, ‘a’, true, false, “Hello World!”
- Variable - a container that stores a value
  - num, flag, name
- Naming convention for variables
  - Variable names should begin with a lowercase letter
  - You can use numbers, underscores, and dollar signs
  - Watch out for reserved words like String, int, boolean

Java has 8 Primitive Types

- **The 4 primitives you will use:**
  - **int** **for integers**
  - **double** **for decimal numbers**
  - **boolean** **for true/false**
  - **char** **for characters**

- The 4 primitives you probably will not use:
  - float
  - byte
  - short
  - long

Note: Strings are NOT primitives!

Examples of int, double, boolean, char, String

```
int num = 3743;
```

```
double num2 = 7.3;
```

```
boolean flag = false;
boolean flag2 = true;
```

```
char c1 = 'l';
char c2 = 'a';
char c3 = 'b';
```

```
String name = "Jesse";
```

Type Casting a Primitive to another Primitive

```
type variableName = (newType) value;
```

```
double myFloat = 3.7;
int num = (int) myFloat;
// num is 3
```

- Value - a piece of information or data
  - 353, 9.3, ‘a’, true, false, “Hello World!”
- Variable - a container that stores a value
  - num, flag, name
- Naming convention for variables
  - Variable names should begin with a lowercase letter
  - You can use numbers, underscores, and dollar signs
  - Watch out for reserved words like String, int, boolean
- You cannot use parentheses to type cast Strings to primitives!

Type Casting a String to a Primitive

```
int num = Integer.valueOf(myString);
```

```
double num2 = Double.valueOf(myString);
```

```
boolean flag = Boolean.valueOf(myString);
```

- Strings are special variables, so they need to use a function called valueOf() in order to cast Strings to primitives
  - Make sure you use the capitalized “Integer” or “Double” so that you can call the function valueOf()

Comparison and Logical Operators

	Operator	Name	Description	Example
Comparison operators	==	Equal to	not to be confused with =	x == y
	!=	Not equal	not to be confused with !	x != y
	>	Greater than		x > y
	<	Less than		x < y
	>=	Greater than or equal to		x >= y
	<=	Less than or equal to		x <= y
Logical operators	&&	And	true if both are true	x > 0 && x < 5
	||	Or	true if at least one is true	x == 0 || x == 1
	!	Not	reverses, false if it is true	!(x > 0)

String methods

```
String s = "Eclipse";
```

s.length()	returns an int, the length of the String
s.charAt(index)	returns a char at the given index
s.equals(s2)	returns true if s is the same String as s2
s.contains(s2)	returns true if s contains s2
s.toUpperCase()	returns "ECLIPSE"
s.toLowerCase()	returns "eclipse"
s.isEmpty()	returns true if s is an empty String ""
s.split(separator)	returns a String array of substrings
s.substring(a,b)	returns a part of the String

More on Strings: https://www.w3schools.com/java/java_ref_string.asp

if statements

```
if (condition) {
    // block of code to be executed
}
```

- if statements execute a block of code if the condition is true
- If the condition is false, the block of code will NOT run!

```
String password = "GoBl@zers";
String input = "WarE@gles";
if (input.equals(password)) {
    // grant user access
}
```

else and else if

else if runs a block of code if the 2nd condition is true and the 1st condition is false

else runs a block of code if all conditions are false

- else statements do not have a condition

```
if (condition) {
    // block of code for 1st condition
} else if (condition 2) {
    // block of code for 2nd condition
} else {
    // block of code to be executed
    // if all other conditions fail
}
```

multiple else ifs

- You can have multiple else if statements in the chain

- If one of the conditions are true, run that block of code and then exit the entire if else chain

```
if (condition) {
    // block of code
} else if (condition 2) {
    // block of code
} else if (condition 3) {
    // block of code for 3rd condition
} else {
    // block of code to be executed
    // if all other conditions fail
}
```

Notice the difference?

```
if (condition) {
    // block of code
} if (condition 2) {
    // block of code
} if (condition 3) {
    // block of code
} else {
    // block of code
}
```

```
if (condition) {
    // block of code
} else if (condition 2) {
    // block of code
} else if (condition 3) {
    // block of code for 3rd condition
} else {
    // block of code to be executed
    // if all other conditions fail
}
```

switch

- switch statements executes one code block from many code blocks
- break keyword stops execution of the the code block
- default keyword specifies code to run if none of the cases match

```
int day = 2;
switch (day) {
    case 1:
        System.out.println("Mon");
        break;
    case 2:
        System.out.println("Tues");
        break;
    default:
        System.out.println("Wed");
}

// prints "Tues"
```

i++ is used to increment

- i++ is “syntactic sugar” for incrementing a variable by plus 1

```
i++;
i = i + 1;
```

- It is the exact same thing as i = i + 1

```
count++;
count = count + 1;
```

i-- is used to decrement

- i++ is “syntactic sugar” for decrement a variable by minus 1

```
i--;
i = i - 1;
```

- It is the exact same thing as i = i - 1

```
count--;
count = count - 1;
```

while loop

- while loops continually loop through a block of code as long as the condition is true
- iterator represents an initial value that changes according to the updater
- The code on the right prints 0, 1, 2, 3

```
iterator;
while (condition) {
    // some block of code
    updater;
}
```

```
int i = 0;
while (i < 4) {
    System.out.println(i);
    i++;
}
```

for loop

- for loops loop as long as the condition is true
- iterator represents an initial value that changes according to the updater
- The code on the right prints 0, 1, 2, 3

```
for (iterator; condition; updater) {
    // some block of code
}
```

```
for (int i = 0; i < 4; i++) {
    System.out.println(i);
}
```

Structure of a for loop and while loop

for loop

```
for (iterator; condition; updater) {
    // some block of code
}
```

while loop

```
iterator;
while (condition) {
    // some block of code
    updater;
}
```

- Every for loop can be rewritten as a while loop
- Every while loop can be rewritten as a for loop

for loop

```
for (int i = 0; i < 4; i++) {
    System.out.println(i);
}
```

while loop

```
int i = 0;
while (i < 4) {
    System.out.println(i);
    i++;
}
```

Both of these do the exact same thing: prints 0, 1, 2, 3

- Use for loops when you know exactly how many times to loop
- Usually we just stick with while loops, but for loops can get the same job done in a more readable way

break

- break keyword stops further execution of a loop
  - Used in for loops, while loops, switch blocks
- Example: this code will attempt to loop 10 times, but will kill the loop as soon as i is equal to 4

```java
for (int i = 0; i < 10; i++) {
    if (i == 4) {
        break;
    }
    System.out.println(i);
}
```

```
Console
<terminated> Str [
0
1
2
3
```

continue

- The continue keyword ends the current iteration of a loop and continues on to the next iteration
- Example: this code will loop 6 times, but will “skip” the iteration where i is equal to 4

```
for (int i = 0; i < 6; i++) {
    if (i == 4) {
        continue;
    }
    System.out.println(i);
}
```

Console

<terminated> Str

```
0
1
2
3
5
```