CS 200 - Programming I: Programming Process

Marc Renault

Department of Computer Sciences
University of Wisconsin – Madison

Fall 2019

TopHat Sec 3 (1:20 PM) Join Code: 682357
TopHat Sec 4 (3:30 PM) Join Code: 296444

WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON

Computational Thinking

Problem Solving

Art of Programming

Abstraction

- Can you see the essential parts of the problem?
- What are the outputs? inputs? their relationship?
- Can you summerize the problem at a high level?
- What are the different components of the solution?

Problem Solving

Art of Programming

Abstraction

- Can you see the essential parts of the problem?
- What are the outputs? inputs? their relationship?
- Can you summerize the problem at a high level?
- What are the different components of the solution?

Computational Thinking

- Think like a computer.
- What is the sequence of actions need to accomplish the task?
- Start with *pseudocode*.

Thinking Like a Computer

Programming is like Designing a Recipe

Chocolate Chip Cookies

Ingredients:

- 227g (1 cup) butter, softened
- 200g (1 cup) sugar
- 105g (½ cup) brown sugar
- 2 eggs
- 2 tsp vanilla
- 250g (2 cups) all-purpose flour
- 1 tsp soda
- 1 pinch salt
- 1 ½ cups of chocolate chips

Instructions:

1. Beat butter, sugars, eggs and vanilla until light and fluffy.
2. Add flour, soda, and salt; blend well.
3. Add chips.
4. Drop from a teaspoon 2 inches apart.
5. Bake 190°C for 9 min.

Original: 375°F for 10 min.
Convection: 325°F for 9 – 10 min.

Thinking Like a Computer

Programming is like Designing a Recipe

Chocolate Chip Cookies

Ingredients:

- 227g (1 cup) butter, softened
- 200g (1 cup) sugar
- 105g (½ cup) brown sugar
- 2 eggs
- 2 tsp vanilla
- 250g (2 cups) all-purpose flour
- 1 tsp soda
- 1 pinch salt
- 1 ½ cups of chocolate chips

Instructions:

1. Beat butter, sugars, eggs and vanilla until light and fluffy.
2. Add flour, soda, and salt; blend well.
3. Add chips.
4. Drop from a teaspoon 2 inches apart.
5. Bake 190°C for 9 min.

Original: 375°F for 10 min.
Convection: 325°F for 9 – 10 min.

Human Thinking

Thinking Like a Computer

Programming is like Designing a Recipe

Chocolate Chip Cookies

Ingredients:

- 227g (1 cup) butter, softened
- 200g (1 cup) sugar
- 105g (½ cup) brown sugar
- 2 eggs
- 2 tsp vanilla
- 250g (2 cups) all-purpose flour
- 1 tsp soda
- 1 pinch salt
- 1 ½ cups of chocolate chips

Instructions:

1. Beat butter, sugars, eggs and vanilla until light and fluffy.
2. Add flour, soda, and salt; blend well.
3. Add chips.
4. Drop from a teaspoon 2 inches apart.
5. Bake 190°C for 9 min.

Original: 375°F for 10 min.
Convection: 325°F for 9 – 10 min.

Computer Thinking

Thinking Like a Computer

Programming is like Designing a Recipe

Chocolate Chip Cookies

Ingredients:

- 227g (1 cup) butter, softened
- 200g (1 cup) sugar
- 105g (½ cup) brown sugar
- 2 eggs, beaten
- 2 tsp vanilla
- 250g (2 cups) all-purpose flour
- 1 tsp soda
- 1 pinch salt
- 1 ½ cups of chocolate chips

Instructions:

1. Beat butter, sugars, eggs and vanilla until light and fluffy.
2. Add flour, soda, and salt; blend well.
3. Add chips.
4. Drop from a teaspoon 2 inches apart.
5. Bake 190°C for 9 min.

Original: 375°F for 10 min.
Convection: 325°F for 9 – 10 min.

Computer Thinking

Thinking Like a Computer

Programming is like Designing a Recipe

Chocolate Chip Cookies

Ingredients:

- 227g (1 cup) butter, softened
- 200g (1 cup) sugar
- 105g (½ cup) brown sugar
- 2 eggs, beaten
- 2 tsp vanilla
- 250g (2 cups) all-purpose flour
- 1 tsp soda
- 1 pinch salt
- 1 ½ cups of chocolate chips

Instructions:

1. Beat butter, sugars, eggs and vanilla until light and fluffy.
2. Add flour, soda, and salt; blend well.
3. Add chips.
4. Drop from a teaspoon 2 inches apart.
5. Bake 190°C for 9 min.

Original: 375°F for 10 min.
Convection: 325°F for 9 – 10 min.

Computer Thinking

Exercise

Calculating the area and circumference of a circle.

1. What are the inputs? outputs? their relationship?
2. Pseudocode it!
3. Code it!

Explaining and Tracing

Explaining vs Tracing

Explaining

Summerize and provide a high-level explanation of what the code does in plain English.

TopHat Q1

Explain the `Circle.java` code:

a. Calculates the area and the circumference of a circle.

b. Reads a radius as input from the user and outputs the the area and the circumference of the corresponding circle.

c. Creates a double variable called rad. Initializes it as 0. Prompts the user for a radius and stores it in rad. Outputs "Area:"; calculates the area of a circle of radius rad and outputs it. Outputs "Circumference:"; calculates the circumference of a circle of radius rad and outputs it.

Explaining vs Tracing

Explaining

Summerize and provide a high-level explanation of what the code does in plain English.

Tracing

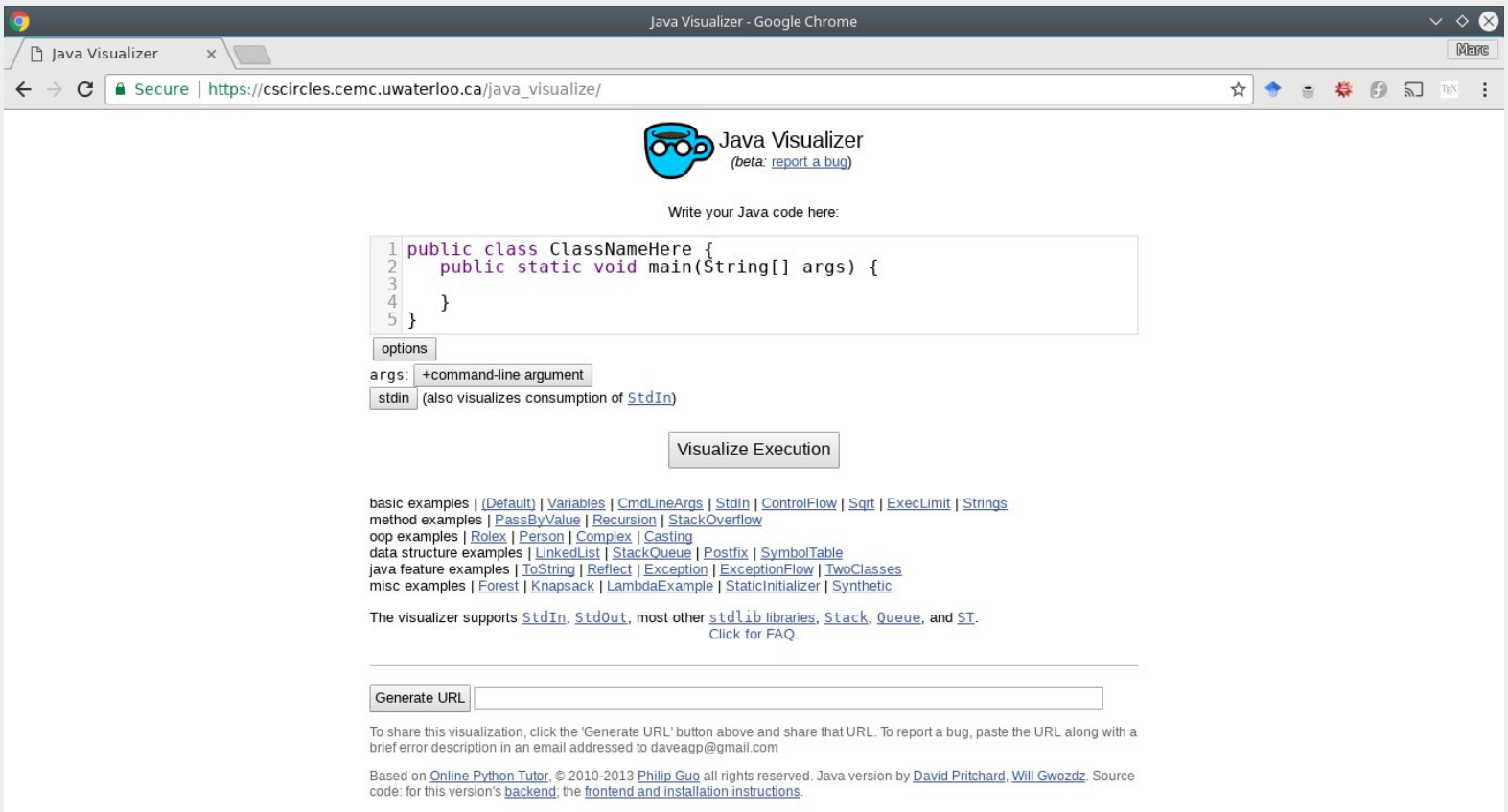
Run the code as computer does.

- Put pen to paper.
- Write down the active variables and their values.
- Update them as they change as you mentally walk through the statements sequentially.

Tool for Tracing

Java Visualizer

- `https://cscircles.cemc.uwaterloo.ca/java_visualize/`



Edit-Compile-Run Cycle

Edit

Writing some source code (set of instructions) in plain text.

Editors

- Text Editors:

vi

Edit

Writing some source code (set of instructions) in plain text.

Editors

- Text Editors:


vi

- Integrated Development Environment (IDE):

Compile

Build your code so that it can be run on a computer.
Often compilers build an *executable* that can be run *natively* on the *target platform*.

Compile

Build your code so that it can be run on a computer.
Often compilers build an *executable* that can be run *natively* on the *target platform*.

`javac` – The *Java* compiler

- `javac` produces a *bytecode* file that needs to be run in a *virtual machine*.

Compile

Build your code so that it can be run on a computer.
Often compilers build an *executable* that can be run *natively* on the *target platform*.

`javac` – The *Java* compiler

- `javac` produces a *bytecode* file that needs to be run in a *virtual machine*.
- *javac* produces a bytecode file with a `.class` file extension.

Compile

Build your code so that it can be run on a computer.
Often compilers build an *executable* that can be run *natively* on the *target platform*.

`javac` – The *Java* compiler

- `javac` produces a *bytecode* file that needs to be run in a *virtual machine*.
- *javac* produces a bytecode file with a `.class` file extension.
- Example:

  ```
  javac Circle.java
  ```

  produces a `Circle.class` *bytecode* file.

Run

Run or execute your code.

java

- The `.class` bytecode file produced by `javac` needs to be run in the Java virtual machine.

Run

Run or execute your code.

java

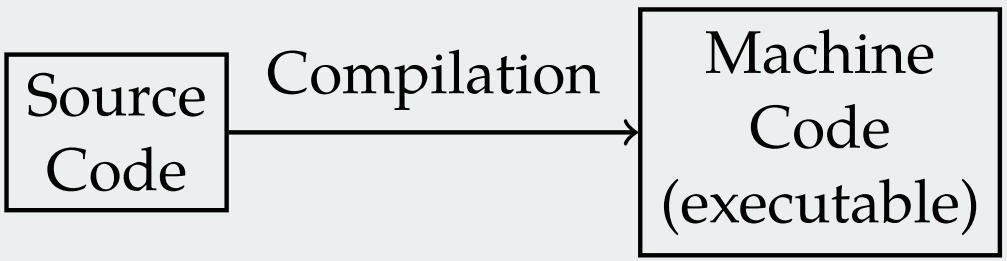
- The `.class` bytecode file produced by `javac` needs to be run in the Java virtual machine.
- Example:

  ```
  java Circle
  ```

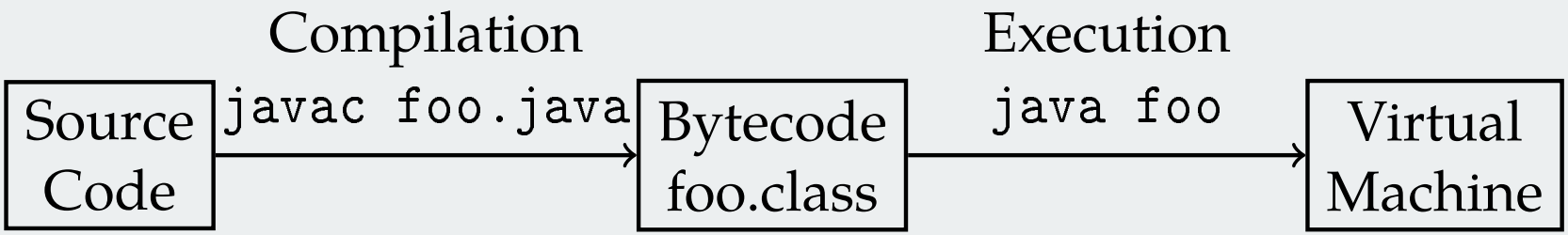
  runs the Circle file.

Virtual Machine

The Classic Model: Native executable



The Java Model: Virtual Machine



TopHat Q2

I have just written some Java code in a file SomeCode.java. How do I compile and run it?

a. java SomeCode
b. javac SomeCode.class
c. javac SomeCode.java
d. java SomeCode.java
e. javac SomeCode
f. java SomeCode.class

Java Initiation

Basic Output

String Literals

- Double quotes ("):

```
"I am a string literal"
```

Basic Output

String Literals

- Double quotes ("):
  `"I am a string literal"`
- Concatenation (+):
  `int a = 10; System.out.println("A " + a);`

Basic Output

String Literals

- Double quotes ("):
  `"I am a string literal"`
- Concatenation (+):
  `int a = 10; System.out.println("A " + a);`
- Newline escape character(\n):
  `"First line\nSecond line"`

Basic Output

String Literals

- Double quotes ("):
  `"I am a string literal"`
- Concatenation (+):
  `int a = 10; System.out.println("A " + a);`
- Newline escape character(\n):
  `"First line\nSecond line"`

Printing to the Console

- Print a string:
  `System.out.print("Does not append a new line");`
- Print a string with a newline:
  `System.out.println("Appends a new line");`

TopHat Q3

What is the output of:

```
System.out.println("5 and 5 = " + 5 + 5);
```

Type the output.

Basic Input

Using the `Scanner`

- Include the library at the top of the file:
  ```
  import java.util.Scanner;
  ```
- Create an *instance* of a `Scanner` *object*:
  ```
  Scanner sc = new Scanner(System.in);
  ```
- Reading an integer:
  ```
  int anInt = sc.nextInt();
  ```

Comments and Whitespace

Comments

- Ignored by the compiler.
- Written by the programmer to explain the code.
- Single-line (//)

```
// Single line comment
```

- Multi-line (/* */)

```
/**
 * Multi-line comment
 */
```

Comments and Whitespace

Comments

- Ignored by the compiler.
- Written by the programmer to explain the code.
- Single-line (//)

```
// Single line comment
```

- Multi-line (/* */)

```
/**
 * Multi-line comment
 */
```

Whitespace

- Mostly ignored by the compiler.
- Good use of white space makes code easier read!

Compiler Errors and Warnings

Compilation Errors

- Prevents compilation
- Syntax errors

Compiler Errors and Warnings

Compilation Errors

- Prevents compilation
- Syntax errors
- Examples:
  - missing ;
  - typo (variable name, keyword)
  - missing braces

Compiler Errors and Warnings

Compilation Errors

- Prevents compilation
- Syntax errors
- Examples:
  - missing ;
  - typo (variable name, keyword)
  - missing braces

Warnings

- Warnings don't stop the compilation process.

Compiler Errors and Warnings

Compilation Errors

- Prevents compilation
- Syntax errors
- Examples:
  - missing ;
  - typo (variable name, keyword)
  - missing braces

Warnings

- Warnings don't stop the compilation process.
- Good practice to write programs that compile without warnings.

Compiler Errors and Warnings

Compilation Errors

- Prevents compilation
- Syntax errors
- Examples:
  - missing ;
  - typo (variable name, keyword)
  - missing braces

Warnings

- Warnings don't stop the compilation process.
- Good practice to write programs that compile without warnings.
- For even stricter compilation, use `-Xlint`:

```
javac -Xlint Foo.java
```

Computer

Basic Machine Architecture

"This is a computer."

von Neumann Architecture

- von Neumann proposed this architecture in 1945.
- Consists of:
  - a processing unit,
  - memory,
  - input devices, and
  - output devices.

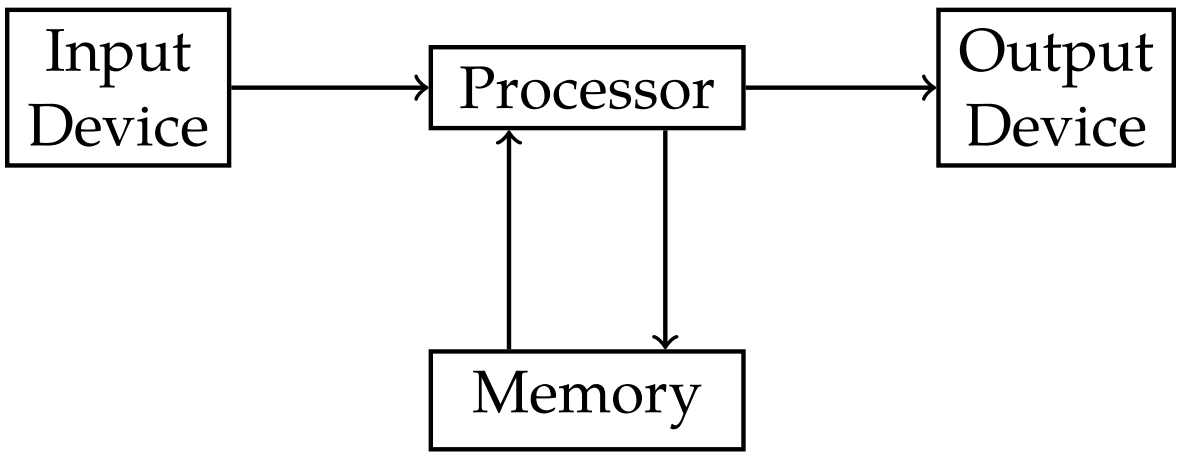
Basic Machine Architecture

John von Neumann

von Neumann Architecture

- von Neumann proposed this architecture in 1945.
- Consists of:
  - a processing unit,
  - memory,
  - input devices, and
  - output devices.

Basic View of a Computer



Further Reading

COMP SCI 200: Programming I
zyBooks.com, 2015.
zyBook code:
WISCCOMPSCI200Fall2019

- Chapter 1. Programming Process

Appendix

References

Image Sources I

http://www.eclipse.org/

https://www.gnu.org/software/emacs/emacs.html

http://packerville.blogspot.ca/2010/05/gentlemen-this-is-football.html

https://en.wikipedia.org/wiki/Microsoft_Notepad

http://2dvocabularynetwokr78.blogspot.fr/

Image Sources II

http://www.lanl.gov/history/atomicbomb/images/NeumannL.GIF

https://brand.wisc.edu/web/logos/

https://commons.wikimedia.org/w/index.php?curid=1228427

https://commons.wikimedia.org/w/index.php?curid=57649239

http://www.zybooks.com/