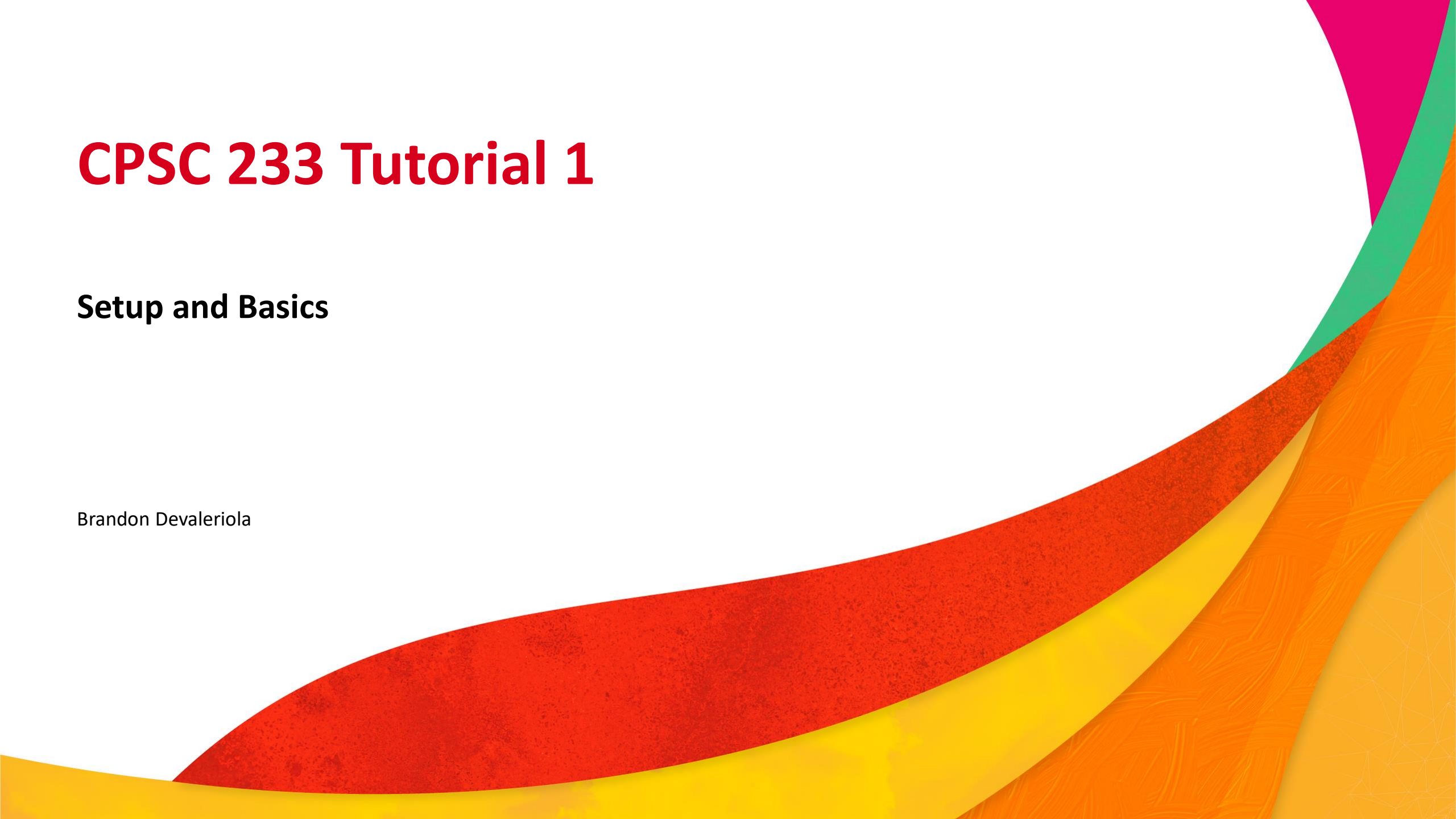


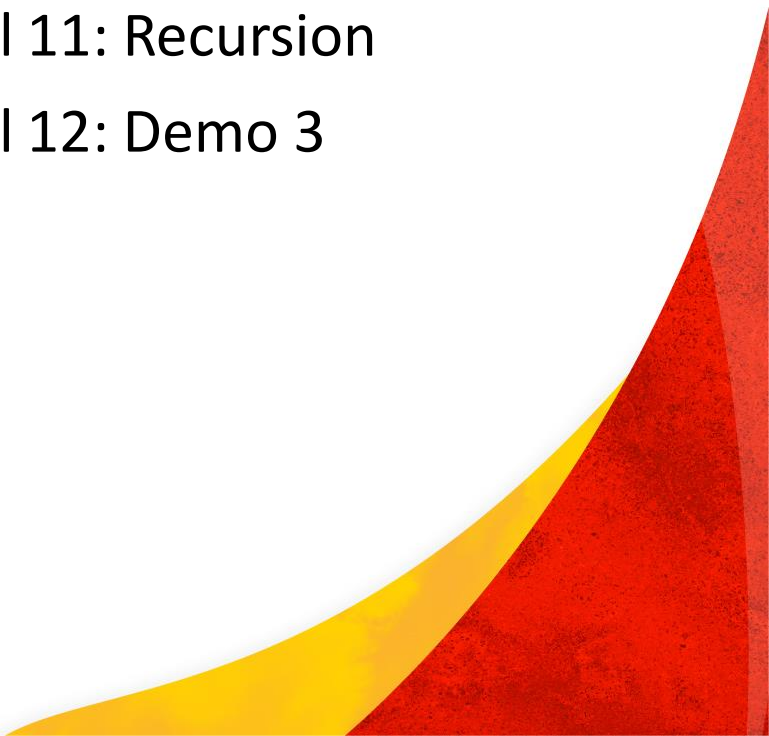
CPSC 233 Tutorial 1

Setup and Basics

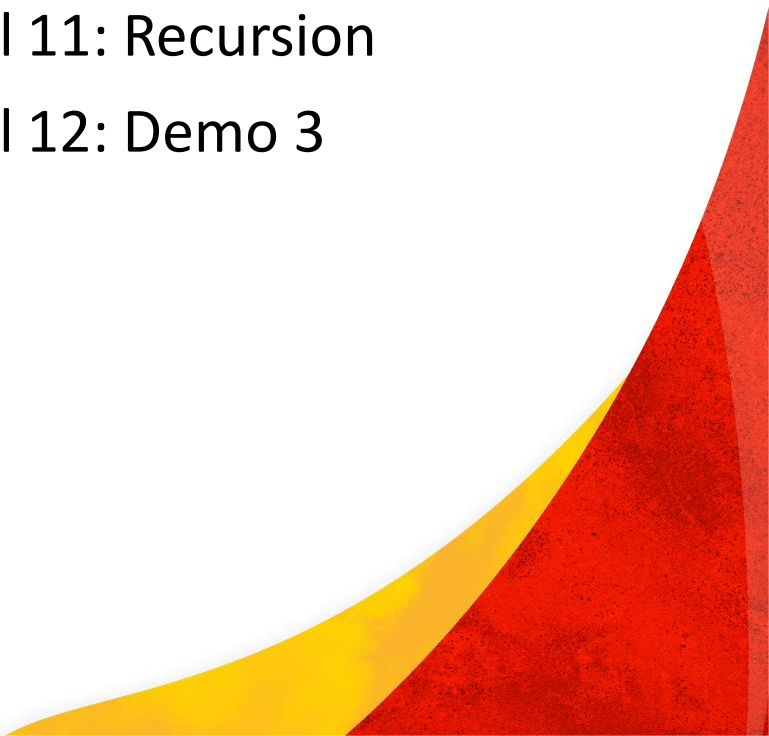
Brandon Devaleriola



Semester Tutorials Overview (tentative)

- Tutorial 1: Setup and Basics
 - Tutorial 2: Flow of Control and Program Structure
 - Tutorial 3: Program Structure and Git
 - Tutorial 4: JUnit and A1 help
 - Tutorial 5: Classes and System Utils
 - Tutorial 6: Demo 1
 - Tutorial 7: Inheritance and A2 help
 - Tutorial 8: Interfaces and JavaFX
 - Tutorial 9: Demo 2
 - Tutorial 10: JavaFX and A3 help
 - Tutorial 11: Recursion
 - Tutorial 12: Demo 3
- 

Semester Tutorials Overview (tentative)

- Tutorial 1: Setup and Basics
 - Tutorial 2: Flow of Control and Program Structure
 - Tutorial 3: Program Structure and Git
 - Tutorial 4: JUnit and A1 help
 - Tutorial 5: Classes and System Utils
 - Tutorial 6: Demo 1
 - Tutorial 7: Inheritance and A2 help
 - Tutorial 8: Interfaces and JavaFX
 - Tutorial 9: Demo 2
 - Tutorial 10: JavaFX and A3 help
 - Tutorial 11: Recursion
 - Tutorial 12: Demo 3
- 

Tutorial 1 Overview

- Part 1
 - Introduction
 - Basic tutorial information and rules
 - Icebreaker!
 - Setup
 - JetBrains education license and Install IntelliJ
 - Quick review
 - Memory and Transistors
 - Binary and Bytes
 - Java Primitive Types
- Break (10 minutes)
- Part 2
 - Class/static void main
 - Scanner library for keyboard input
 - Changing types
 - Doing math
 - Printing and formatting output

Tutorial 1 Part 1



Tutorial 1 – Part 1: Overview

- Part 1
 - Introduction
 - Basic tutorial information and rules
 - Icebreaker!
 - Setup
 - JetBrains education license and Install IntelliJ
 - Quick review
 - Memory and Transistors
 - Binary and Bytes
 - Java Primitive Types



Tutorial 1 - Part 1: Introduction



Brandon Devaleriola (de-valerie-oh-la)

brandon.devaleriola@ucalgary.ca

- 12+ years of software development experience in industry
- L7 Principal Software Engineer
- Previous employers: Lightspeed Commerce, Dyspatch, Unity Technologies, VizworX
- Previous contractor positions: Electronic Arts, Lego, Softbank, Chevron
- <https://www.linkedin.com/in/brandon-devaleriola/>
- Working on a mid-career masters in computer science with a specialization in software engineering

Tutorial 1 – Part 1: Overview

- Part 1
 - Introduction
 - Basic tutorial information and rules
 - Icebreaker!
 - Setup
 - JetBrains education license and Install IntelliJ
 - Quick review
 - Memory and Transistors
 - Binary and Bytes
 - Java Primitive Types

Tutorial 1 - Part 1: Basic tutorial information and rules

- I don't know everything about Java you may know more than me, or I may miss something. Please correct me if I am wrong, I won't mind
- I'm bad at remembering name, I will try to but it's not personal if I don't
- TA and Tutorials are for:
 - Working on live examples during tutorial that relate to content presented in earlier lectures
 - Asking questions / getting help on assignments
 - Getting help when you have a problem related to the class but not directly code related. For example, like having issues with SSH keys, or you did a command in git, and you don't know how to revert it
- I am most responsive from 9AM-5PM weekdays and will try to respond within a two hours during that time
- Outside of those hours will usually result in a next day response (I may check on weekends during a coming assignment due dates, but I make no guarantees)
- I also have classes and research work, so I would appreciate as few LMGTFY questions if possible
- I get many emails, and there is a chance I can miss your email. I have a filter inbox for TA emails. Help me, help you, by adding the following to your email subject line: "TA 233 – [topic]"
 - Where [topic] is a summary of the issue you are having or a question.
 - In the email body please add screenshots, and additional information I can use to help you
 - Example: "TA 233 – IntelliJ license issue"
- We have a class discord, I check it, but please do not use Discord as the primary way to contact me for anything important. Please email me instead

Tutorial 1 – Part 1: Overview

- Part 1
 - Introduction
 - Basic tutorial information and rules
 - Icebreaker!
 - Setup
 - JetBrains education license and Install IntelliJ
 - Quick review
 - Memory and Transistors
 - Binary and Bytes
 - Java Primitive Types



Tutorial 1 - Part 1: Icebreaker!

- Icebreaker!



Tutorial 1 - Part 1: Icebreaker!

- Icebreaker!
- I want you to do this task:
 1. Login to your @ucalgary.ca email
 2. Create a new email to: **brandon.devaleriola@ucalgary.ca**
 3. In the subject line add the text: “TA 233 – introduction”
 4. In the body of the email add some information you are willing to share about yourself:
 - where you grew up
 - any accessibility needs you may have (it’s the best time to tell me)
 - maybe something interesting about you
 - why you want to learn how to program
 - maybe some video games you like
 - and any questions you have for me
 - a dank meme

Tutorial 1 - Part 1: Icebreaker!

- Thanks for the email introduction email!
- Now that you have emailed me, you now have no excuse to say you didn't know how to contact me



Tutorial 1 – Part 1: Overview

- Part 1
 - Introduction
 - Basic tutorial information and rules
 - Icebreaker!
 - Setup
 - [JetBrains education license and Install IntelliJ](#)
 - Quick review
 - Memory and Transistors
 - Binary and Bytes
 - Java Primitive Types



Tutorial 1 - Part 1: Setup

Setup local IDE

- Login and out of lap workstations
- If you have your own laptop it's recommended to set that up for this class. Here is how we will do that:
 1. Create JetBrains account using your @ucalgary.ca email and request student license
 2. Download and install IntelliJ Ultimate and ensure license is enabled

Tutorial 1 – Part 1: Overview


- Part 1
 - Introduction
 - Basic tutorial information and rules
 - Icebreaker!
 - Setup
 - JetBrains education license and Install IntelliJ
 - Quick review
 - Hardware
 - Memory
 - Binary and Bytes
 - Java Primitive Types

Tutorial 1 - Part 1: Quick Review -> Hardware

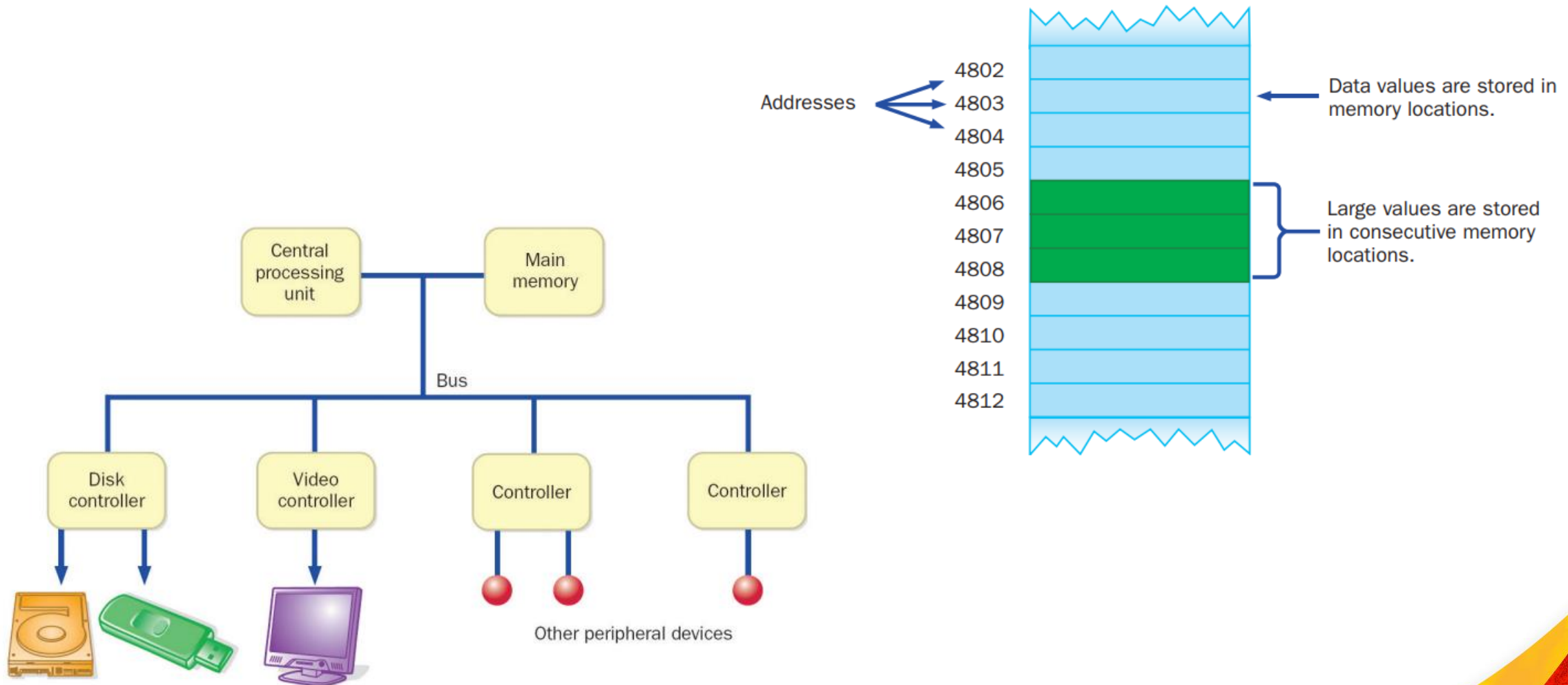
Understand how computer hardware relates to primitive types

- Memory: Volatile(RAM) and Non-volatile (HDD, SSD, M.2, ROM)
 - Binary and Bytes
- 

Tutorial 1 - Part 1: Quick Review -> Hardware -> Memory

- Main memory is made up of billions of transistors
 - Transistors are a type of semiconductor that is used for fast-switching of electronic gates using electricity. We use transistors and their possible states to represent binary data. 0 represents off, and 1 represents on.
 - Random Access Memory (RAM) is series of consecutive locations. Associated with each memory location is a unique number called an address.
- 

Tutorial 1 - Part 1: Quick Review -> Hardware -> Memory

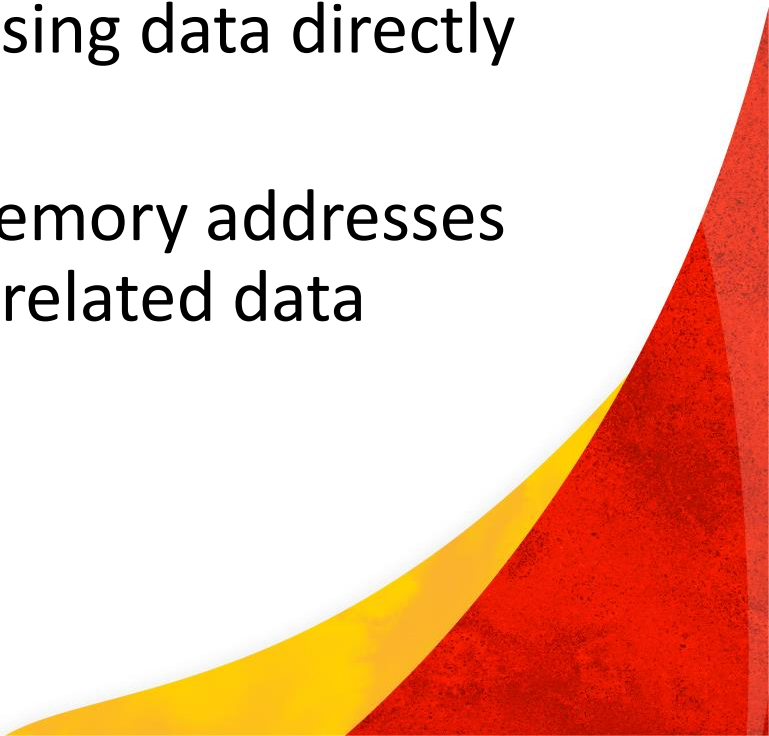


Tutorial 1 - Part 1: Quick Review -> Binary and Bytes

1 bit 2 items	2 bits 4 items	3 bits 8 items	4 bits 16 items	5 bits 32 items
0	00	000	0000	00000
1	01	001	0001	00001
	10	010	0010	00010
	11	011	0011	00011
		100	0100	00100
		101	0101	00101
		110	0110	00110
		111	0111	00111
			1000	01000
			1001	01001
			1010	01010
			1011	01011
			1100	01100
			1101	01101
			1110	01110
			1111	01111

- Binary is a number system that is only two digits: 0 and 1
- Binary is the lowest level of data storage, transmission, and manipulation in computers.
- Each binary digit is a Bit, and 8 Bits is a Byte

Tutorial 1 - Part 1: Quick Review -> Variable Basics

- Variables are a way to store different types of data with human readable name
 - Variables are an abstraction layer of high-level languages that allow programmers to use simple names instead of accessing data directly from memory addresses
 - Variables allow us to use contiguous “chunks” of memory addresses for more complex data (objects) where we wanted related data stored.
- 

Tutorial 1 - Part 1: Quick Review -> Variable Basics


- Java has 9 Primitive types with 3 subcategories:
 - Textual Primitives (byte, char)
 - Boolean Primitives (boolean, null)
 - Numeric Primitives (short, int, long, float, double) – shown in the chart below

Type	Storage	Min Value	Max Value
byte	8 bits	−128	127
short	16 bits	−32,768	32,767
int	32 bits	−2,147,483,648	2,147,483,647
long	64 bits	−9,223,372,036,854,775,808	9,223,372,036,854,775,807
float	32 bits	Approximately $-3.4\text{E}+38$ with 7 significant digits	Approximately $3.4\text{E}+38$ with 7 significant digits
double	64 bits	Approximately $-1.7\text{E}+308$ with 15 significant digits	Approximately $1.7\text{E}+308$ with 15 significant digits


Tutorial 1 - Part 2



Tutorial 1 - Part 2: Overview

- Class/static void main
 - Scanner library for keyboard input
 - Changing types
 - Doing math
 - Printing and formatting output
- 

Tutorial 1 - Part 2: Overview

- Class/static void main
 - Scanner library for keyboard input
 - Changing types
 - Doing math
 - Printing and formatting output
- 

Tutorial 1 - Part 2: Class/static void main

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

Let's break down each part:

1. **public**: This is an access modifier that means the method is accessible everywhere, including from outside the class it's defined in. It needs to be `public` so that the JVM can call it.
2. **static**: This means the method belongs to the class itself, rather than to instances (objects) of the class. It needs to be `static` because the JVM calls this method without creating an instance of the class first.
3. **void**: This is the return type of the method. `void` means the method doesn't return anything. The `main` method doesn't need to return anything because it's not meant to be called by your code; it's meant to be called by the JVM.
4. **main**: This is the name of the method. The JVM looks for a method with this exact name to start the application.
5. **String[] args**: This is the parameter of the method. It's an array of `String` objects, which represents any command-line arguments that were passed when the application was started. For example, if you start the application with `java MyApp arg1 arg2`, then `args` will be an array containing `"arg1"` and `"arg2"`.

The body of the `main` method is where you put the code that you want to run when the application starts. This can be anything from a simple "Hello, World!" print statement to a complex application with multiple classes and methods.

Tutorial 1 - Part 2: **Class/static void main**

Exercise 1 – Coding warm-up

Create a traditional hello world program in Java


- Create a class called HelloWorld
- The class has a single method called main()
- main() contains a single print line function call
- The program should run without errors
- the text output should be:

Hello, World!

Tutorial 1 - Part 2: Class/static void main

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

Tutorial 1 - Part 2: Overview

- Class/static void main
 - Scanner Class for input
 - Changing types
 - Doing math
 - Printing and formatting output
- 

Tutorial 1 - Part 2: Scanner Class for input

```
Scanner(InputStream source)
```

```
Scanner(File source)
```

```
Scanner(String source)
```

Constructors: sets up the new scanner to scan values from the specified source.

```
String next()
```

Returns the next input token as a character string.

```
String nextLine()
```

Returns all input remaining on the current line as a character string.

```
boolean nextBoolean()
```

```
byte nextByte()
```

```
double nextDouble()
```

```
float nextFloat()
```

```
int nextInt()
```

```
long nextLong()
```

```
short nextShort()
```

Returns the next input token as the indicated type. Throws

`InputMismatchException` if the next token is inconsistent with the type.

```
boolean hasNext()
```

Returns true if the scanner has another token in its input.

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

Tutorial 1 - Part 2: Scanner Class for input

Exercise 2 – Fuel Efficiency Calculator

Create a Fuel Efficiency Program

- The class has a single method called main()
- main() should use an instance of the Scanner class to accept input
- User should be prompted for Liters of fuel used
- User should be prompted for Kilometers of distance travelled
- Calculate efficiency by consumption of liters per 100 kilometers.
- the text output should be like:

```
Enter liters of fuel used: 60.00
```

```
Enter distance travelled in kilometers: 550.00
```

```
Efficiency is: 10.909090909090908 L/100 km
```


Tutorial 1 - Part 2: Scanner Class for input

```
import java.util.Scanner;

▶ public class FuelEfficiencyCalculator {
    1 usage
    static final double KILOMETERS = 100.00;

▶    public static void main(String[] args) {
        ⚡ Scanner scanner = new Scanner(System.in);
        System.out.print("Enter liters of fuel used: ");
        String fuelUsed = scanner.nextLine();
        double dFuelUsed = Double.parseDouble(fuelUsed);
        System.out.print("Enter distance travelled in kilometers: ");
        String distanceTravelled = scanner.nextLine();
        double ddistanceTravelled = Double.parseDouble(distanceTravelled);
        double distanceBy100km = ddistanceTravelled / KILOMETERS;
        double efficiency = dFuelUsed / distanceBy100km;
        // System.out.println("Efficiency is: " + efficiency + " L/100 km"); // unformatted double
        System.out.format("Efficiency is: %5.2f L/100 km", efficiency);
    }
}
```

Tutorial 1 - Part 2: Overview

- Class/static void main
 - Scanner library for keyboard input
 - Changing types
 - Doing math
 - Printing and formatting output
- 

Tutorial 1 - Part 2: Changing types

Converting Types to other types

- Widening Conversion

From	To
byte	short, int, long, float, or double
short	int, long, float, or double
char	int, long, float, or double
int	long, float, or double
long	float or double
float	double

- Narrowing Conversion

From	To
byte	char
short	byte or char
char	byte or short
int	byte, short, or char
long	byte, short, char, or int
float	byte, short, char, int, or long
double	byte, short, char, int, long, or float

Tutorial 1 - Part 2: Changing types

Ways conversion can happen

- **Assignment Conversion** - occurs when a value of one type is assigned to a variable of another type during which the value is converted to the new type. Only widening conversions can be accomplished through assignment.
 - `money = dollars;`
 - Where `money` is a float and `dollars` is an int
- **Promotion** - occurs automatically when certain operators need to modify their operands to perform an operation. For an operation to occur both types need to be same to compute the result.
 - `float result = sum / count;`
 - `sum` is a floating-point primitive type and `count` is an integer. `count` is promoted to a floating point before the division happens
- **Casting** - Casting explicit and is written in the code, compared to the previous two which are implicit and by the runtime. Casting is the most general form of conversion in Java. If a conversion can be accomplished at all in a Java program, it can be accomplished using a cast.
 - `revolutions = (int) speed;`
 - where `revolutions` is an int and `speed` is a double

Tutorial 1 - Part 2: Changing types

Exercise 3 – Conversion Experiments

Create a Conversion Experiments class


- In the class add 4 multi-line comments following types: *int*, *char*, *double*, *float*
- For each type add 5 experiments (examples) of conversions with different operations that can happen on that type using assignment and different operators.
- Try to list the different ways conversions can happen on that type and include examples narrowing or widening conversions.
- For each experiment add concrete details related to that specific experiment like initial value, operations or assignments, expected resulting type and expected output value after the conversion
- Convert each experiment into working code one line at a time and print the resulting value and the using:

```
System.out.println(resultingValue + " is of type " + ((Object)resultingValue).getClass().getSimpleName());
```

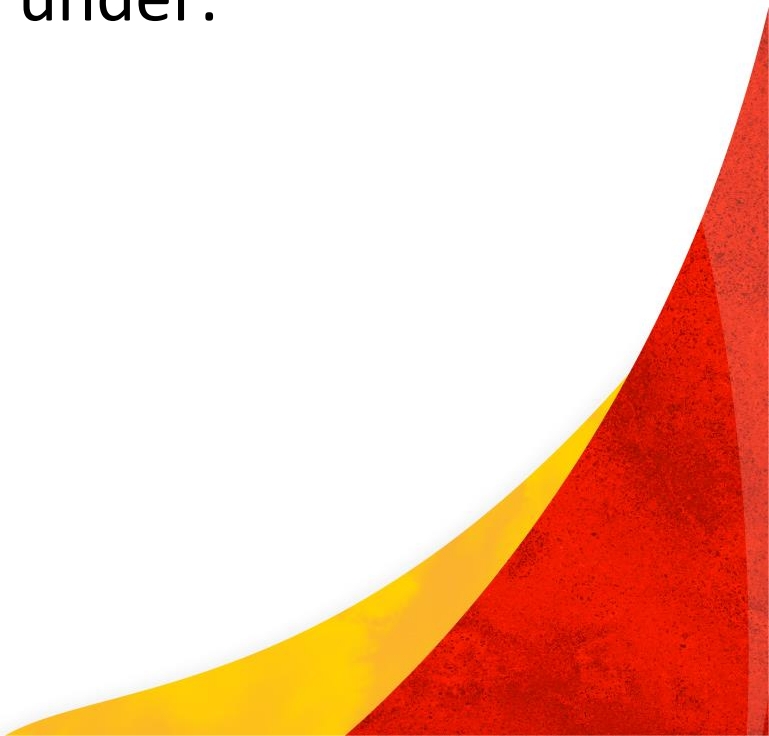

Tutorial 1 - Part 2: Changing types

```
/*  
  int  
  1. widening assignemnt  
    int to float  
    example: money = dollars; Where money is a float and dollars is an int  
    inital value: int dollars = 25; double money;  
    operations:  
    assignments: money variable  
    expect resulting type: double  
    expected resulting value: 25.0  
    description: money should equal 25.0 after assignment  
  2. ....  
  3. ....  
*/
```

Tutorial 1 - Part 2: Overview

- Class/static void main
 - Scanner library for keyboard input
 - Changing types
 - Doing math
 - Printing and formatting output
- 

Tutorial 1 - Part 2: Doing Math

- Operations have different precedence levels in Java
 - The operation precedence can impact when type conversions happen
 - In precedent order basic math operations are classified under:
 - parentheses: ()
 - unary(post): `expr++` `expr--`
 - unary(pre): `++expr` `--expr`
 - cast: (type)
 - multiplicative: `*` `/` `%`
 - additive: `+` `-`
 - assignment: `=` `+=` `-=` `*=` `/=` `%=`
- 

Tutorial 1 - Part 2: Doing Math

Exercise 4 – Relative Temperature Display

Create a Relative Temperature Program

- Create variables for
 - temperature in Celsius
 - wind speed
 - wind chill calculation result(used for feels like in display)
 - wind chill to display (wind chill result minus current temp in display)
 - feels like in Fahrenheit
- Calculate wind chill from wind speed and temperature and conversions
- Example: a windspeed of 12 km/h and temp of -36 the output should be:


Current Temperature: -36 Celsius

Wind Chill: -11

Feels like: -47

Feels like Fahrenheit Equivalen : -57

Tutorial 1 - Part 2: Overview

- Class/static void main
 - Scanner library for keyboard input
 - Changing types
 - Doing math
 - Printing and formatting output
- 

Tutorial 1 - Part 2: Printing and formatting output

- Working with the java System library to display text data
- *System.out.println()* versus *System.out.print()* – cursor position
- Strings are objects not primitive types
- *System.out.format(string, args)* can be used with a *format* string, and second parameter *args* are to be formatted

```
System.out.format("The value of " + "the float variable is " +  
    "%f, while the value of the " + "integer variable is %d, " +  
    "and the string is %s", floatVar, intVar, stringVar);
```

Tutorial 1 - Part 2: Printing and formatting output

Exercise 5 - Create a count down program to print the countdown of a launching rocket and total cost of the flight

- Create variables for rocket fuel price in dollars per liter, distance traveled in meters, time of flight in seconds, and base cost of the rocket.
- Calculate the costs of the flight per second and format the output as dollars with two significant digits using Java's built in print formatting.
- The resulting text output of should be:

Three... Two... One... Zero... Liftoff!

Houston, we have a problem.

Houston, flight total cost was \$19,304,408.42 per second.

Tutorial 1 - Part 2: **Finished**

My solutions for exercises 3, 4, and 5 can be made available at the start of next tutorial



References

- <https://docs.oracle.com/javase/tutorial/java/nutsandbolts/operators.html>
- <https://introcs.cs.princeton.edu/java/11precedence/#:~:text=Operator%20precedence%20specifies%20the%20manner,precedence%20than%20the%20addition%20operator>
- Lewis, J., & Loftus, W. (2017). *Java software solutions : foundations of program design* (Ninth edition.). Pearson.