The Hawk Collective
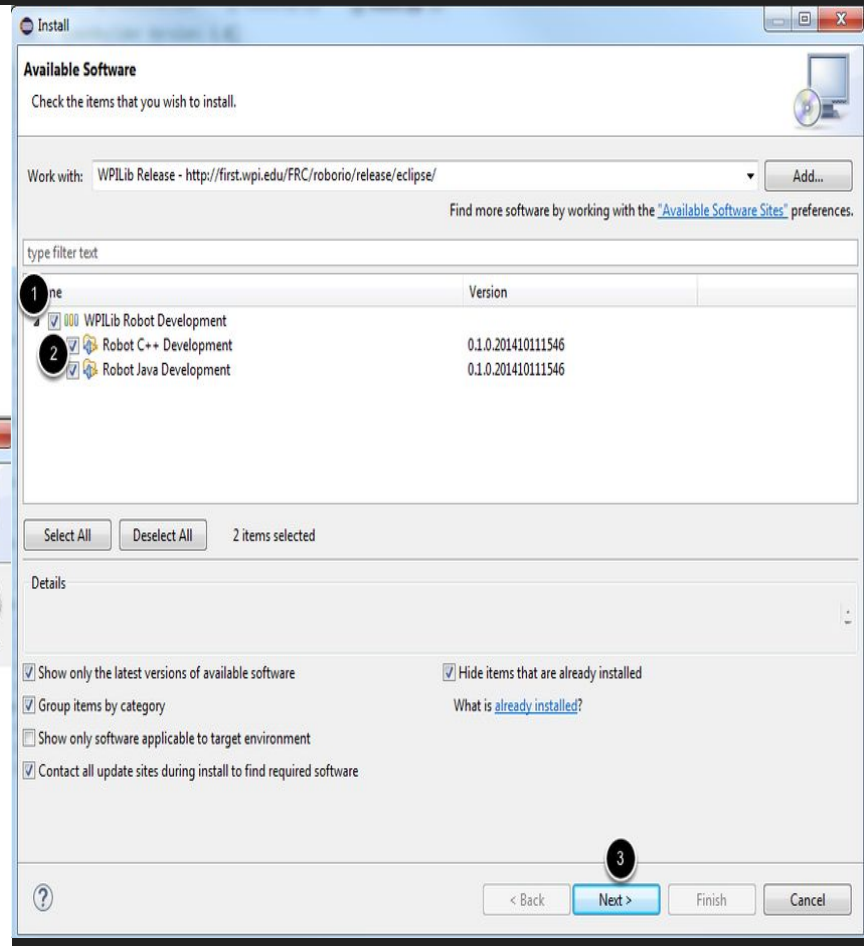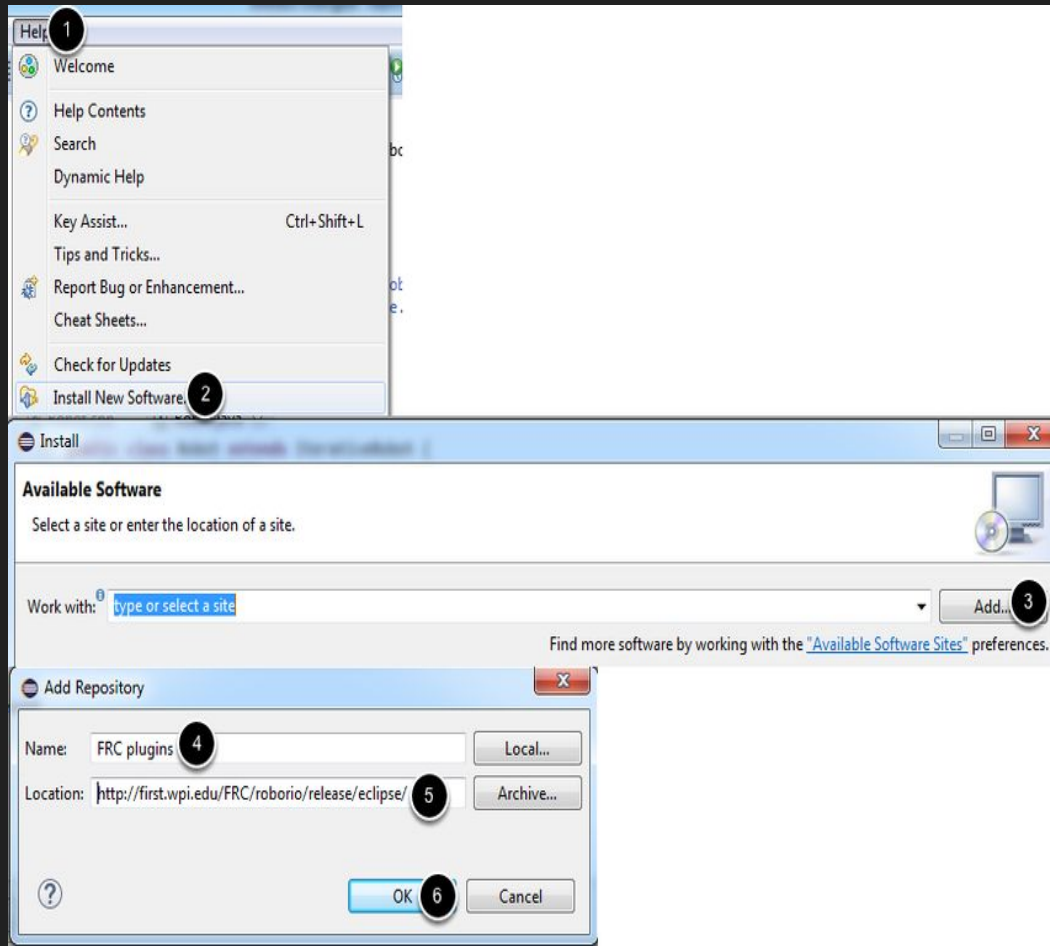
Java Programming Tutorial

Setting Up Your Environment for Java Development

Download and install Eclipse

Download and install the Java Development Kit (JDK)

Installing FIRST Eclipse Plugins

Starting a New Project in Eclipse

To create a new project:

File -> New -> Java Project (Keep default settings)

To create a new class (You'll need at least one of these in each project):

File -> New -> Class (If this is your 'main' class make sure to check this box)

Which method stubs would you like to create?

☑ public static void main(String[] args)

☐ Constructors from superclass

Java Program Execution Flow

Java Programs begin execution at a 'main' method.

The main method will always have the following method header

```java
public static void main(String[] args)
```

Java programs will run each line of code sequentially starting with the first line of the main method

Java Syntax (Grammar)

Java is case-sensitive, 'test' and 'Test' are processed as two completely distinct words

Each line of code ends in a semicolon ';'

By convention, variable and method names are written in what is known as Camel Case. This means that the first word is lowercase, with following words being capitalized (with no spaces between them) e.g. variableName

Curly braces '{}' are used to denote sections of related code

Lines that begin with '//' or blocks of text bookended with '/*' and '*/' are comments and will not affect program execution

Data Types

Data can be stored in a variety of formats within any programming language.

Java's basic or 'primitive' types:

- Int (Integer values)
- Boolean (true/false)
- Double (decimal values)
- Char (character values e.g. 'a')
- Long, short, byte, float (These aren't really that important)

Data can also be stored in Objects and arrays, which will be covered later

Variables

There are two steps to making a variable in Java, Declaration and Initialization

Declaration is simply stating the type of the variable, followed by the name

```java
int variableName;
```

This tells the computer to make a variable of type int (integer) called 'variableName'

Initialization is assigning a starting value to a variable. This can be (but does not have to be) done on the same line as the declaration.

```java
int variableName = 4;
variableName = 4;
```

Constants

Constants are variables that are declared with the 'final' keyword

```java
final int CONSTANT_VALUE = 42;
```

By convention, constants are name in all caps, with underscores '_' used to separate words

Once initialized, constants cannot be changed.

Mathematical Operators

Basic math operators (+, -, *, /) will function as expected including order of operations (When dividing two integers, the result is always rounded down)

A single equals sign '=' is used as the assignment operator. It will make the left side of the statement equal to the right side.

Two equals signs '==' are used to check equality, i.e. is the left side equal to the right side.

In the code below, the first line makes the value of x equal to 5 + 5 (AKA 10), whereas the second line checks to see if x is equal to 5 + 5 (and will evaluate to either true or false)

```
x = 5 + 5;
x == 5 + 5;
```

Mathematical Operators 2

The modulo operator '%' is used to take the remainder from dividing two numbers

```
7 % 3 == 1;
```

The increment operator '++' and decrement operator '--' can be used to increase or decrease the value of a variable by 1

```
int x = 1;
x++;
//x == 2;
x--;
//x == 1;
```

Assignment Operators

You can combine the basic math operators with the assignment operator '=' to change the value of a variable. The statement 'x += 1' is equivalent to 'x = x + 1'

```
int x = 1;
x += 1;
//x == 2;
x *= 3;
//x == 6;
```

```
x /= 2;
//x == 3;
x -= 1;
//x == 2;
x %= 2;
//x == 0
```

Mathematical vs Assignment Operators

Mathematical operators do not change the value of the variables involved, unless an assignment operator is also used

```
int x = 1;
x + 5;
//x == 1
x = x + 5;
//x == 6
```

The Print Method

The below method will print a single line to the standard output, typically the command line

```java
int x = 3;
System.out.println();
System.out.println(3);
System.out.println(x);
```

The above code will have the following output

```
3
3
```

Boolean (Logical) Operators

Boolean operators are those which can affect statements which are evaluated as either true or false

The NOT operator '!' will reverse the true/false state of its operand (what the operator acts on)

```
boolean example = true;
//!example == false
```

```
boolean example = false;
//!example == true
```

The AND operator will evaluate to true if and ONLY if both operands are true

```
boolean example = true;
boolean trueBool = true;
//example && trueBool == true
```

```
boolean example = true;
boolean falseBool = false;
//example && falseBool == false
```

Boolean Operators 2

The OR operator '||' will evaluate to true if either of its operands is true

```
boolean example = false;
boolean trueBool = true;
//example || trueBool == true
```

```
boolean example = false;
boolean falseBool = false;
//example || falseBool == false
```

The not equals operator '!=' will evaluate to true if the operands are not equal to one another

```
boolean example = true;
boolean falseBool = false;
//example != falseBool == true
```

```
int x = 5;
//x != 5 == false
//x != 7 == true
```

Relational Operators

Relational operators (<, >, <=, >=) will work as you would expect when used with numerical values

```
int x = 5;
//x < 5 == false
//x <= 5 == true
//x > 2 == true
//2 > x == false
```

```
int x = 5;
int y = 2;
//x + y < 4 == false
//x + y >= 7 == true
```

Boolean Expressions

A Boolean Expression is any statement which can be evaluated into true or false.
These expressions often chain together multiple boolean operators

```
int x = 5;
int y = 2;
int z = 4;
((x + y > 4) && (z - x != 1))
// true       AND    true  == true
((x / y > 4) || !(z - x == -1))
// false      OR   NOT(true)
// false      OR    false  == false
```

If Statements

If statements are used to make decisions while the program is running.

```
boolean condition = true;
if (condition)
{
    //if condition is true go here
}
```

A boolean expression, or condition, is placed in the '()' and if the expression is true, the block of code is executed. If the condition is false, then the if statement is skipped and the program will resume executing on the line after the closing brace '}'

Else Statements

Else statements are blocks of code which are executed only if the condition of the if statement they are preceded by is false

```
boolean condition = false;
if(condition)
{
    //if condition is true go here
}
else
{
    //if condition is false go here
}
```

Nested if and else statements

If and else statements can be placed within other if and else statements. These are referred to as 'nested' statements

It is important to keep paired braces at the same indentation level so you can easily see where blocks of code begin and end

```java
boolean condition = false;
boolean secondCondition = true;
if(condition)
{
    if(secondCondition)
    {
        //both conditions are true
    }
    else
    {
        //condition is true
        //secondCondition is false
    }
}
else
{
    //condition is false
}
```

A Variable's "Scope"

"Scope" is the part of a program from which a variable is accessible. If you are within the scope, you can see and use the variable, but if you are outside of the scope you cannot interact with the variable in any way.

In general scope is defined by curly braces '{}', so anything that was defined within a set of braces is visible anywhere else within those braces, including within other nested braces

This means that variables which are declared within an if statement cannot be used outside of that if statement

Scope Example

The variable y is defined in scope A, and is therefore usable anywhere within scope A

Variable x is defined in scope B so is only usable within scope B (The red text color means that there is an error as that line is not within scope B)

```java
public static void main(String[] args)
{ //Scope A
    int y = 4;
    if(true)
    { //Scope B
        int x = 7;
        y = x + y;
    }//End Scope B
    y = x + 2;
}//End Scope A
```

Scope B itself is within scope A, therefore anything defined in scope A is usable in scope B (therefore there is no error when using variable y within scope B)

Garbage Collection

When a program reaches the end of a certain scope (end of a loop, if statement, method, etc.) any references to variables which were created in that scope are deleted.

Any variables which do not have any references pointing to them are automatically deleted, this process is known as garbage collection.

```
if (true)
{
    int i = 4;
}
//i has been garbage collected
```

While Loops

```
int x = 1;
while(x < 3) //First condition is checked
{                //is x less than 3
    x++; // If so Add 1 to x
    //Then go back to the beginning of the loop
    //and check the condition again
} //This loop will run twice
```

A while loop is a block of code that will continue to repeat itself 'while' a condition (boolean expression) is true

The condition is checked for 'truthiness' prior to each iteration of the loop

Similarly to an if statement, if the condition is false the block of code within the while loop will never be run

Infinite Loops

It is important to make sure that the condition of a while loop can become false while iterating through the loop.

If the condition cannot become false then the loop will run forever

```
int x = 1;
while(x < 3)
{
    x--; //x is decreasing
    //therefore it will always
    //be less than 3
}
```

For Loops

```
int sum = 0;
for(int i = 0; i <= 10; i++)
{
    sum += i;
    //This sums all integers 0-10
}
```

The for loop is essentially just a while loop with a built in counter. You use a for loop when you know ahead of time exactly how many times you want to go through the loop

The condition of the for loop consists of three statements
- Initialization - This is where you create the variable that will be used to keep track of which iteration of the loop you are on (i.e. first time through the loop i == 1, second time i == 2, etc.)
- Termination - Boolean expression which terminates the loop when false
- Incrementation - increments the counter variable

For/While Loop Example

Both of the below programs do the same thing, one is implemented with a for loop, the other with a while loop

```
int sum = 0;
for(int i = 0; i <= 10; i++)
{
    sum += i;
    //This sums all integers 0-10
}
```

```
int sum = 0;
int i = 0;
while(i <= 10)
{
    sum += i;
    i++;
}
```

Methods

Methods are sequences of code statements which perform a single operation and can be called or invoked by other sections of code

Each method has what is known as a method signature which defines visibility, return type (output), name, and parameters (input)

There are three visibility modifiers in Java

- Public - this is visible to anyone
- Private - this is only visible within the same class
- Protected - this is only visible within this class or subclasses of this class

*Classes will be covered later

Methods 2

The return type for a method can be any primitive data type, array, or class

Methods can also not return a value, in this case the method signature will contain the keyword 'void' in place of a return type

By convention, method names are written in camel case, and should describe what the method does

Similarly to mathematical functions, the parameters (input) for Java methods are specified as comma separated values which are declared within parentheses following the method name. (e.g. f(x, y))

Methods 3

The below code defines a publically visible method 'average' which takes doubles 'a' and 'b' as parameters, and returns a double which is the average of 'a' and 'b'

```java
public double average(double a, double b)
{
    return (a + b) / 2;
}
```

Using Objects and Classes

A 'class' is an entity in programming which represents and describes a type of thing.

An 'object' is a specific instance of a class

The concept of a book is an example of a class, whereas a specific book would be an object.

Classes can have numerous 'fields' which can store variables or objects, as well as methods which perform some operation

By convention, class names are capitalized and object names use camel case

Using Objects and Classes 2

Objects must be declared and initialized just like variables

Every class as what is known as a 'constructor' method which must be called with the 'new' operator to create an object of that class

Constructor methods share the name of class and do not specify a return type (e.g. The constructor method for the String class has the following method signature)

```
public String()
```

The below code demonstrates how to create an Object of the String class

```
String firstObject = new String();
```

Testing Object Equality

When used with objects, the '==' operator compares the memory addresses associated with the objects.

To test whether two objects have equivalent values, you should use the equals() method.

The 'static' Modifier

By default, methods defined in classes need to be run from an object. This is because many methods require data that is stored in the object to properly function

The static modifier is used when a method can be run directly from the class itself and does not need to be attached to an object

Strings

A String is a sequence of characters and can be used to represent text or store data such as a name.

A String object can be created like any other object, using 'new' and calling the constructor, or by assigning it to text within quotes (").

```
//Creates a blank String object
String exampleString = new String();
//reassigning exampleString to be equal to "Example String"
exampleString = new String("Example String");
//Creates a string using a string literal
String literalString = "literal string";
```

Text that is contained within quotes is known as a String literal.

String operators

Strings in Java have some unique behavior when used with common operators.

The '+' operator can be used to concatenate (Append one to the other) two Strings. Similarly the '+=' operator can be used to concatenate and assign strings in one statement.

```java
String helloWorld = "Hello";
//helloWorld == "Hello Wo"
helloWorld = helloWorld + " Wo";
//helloWorld == "Hello World"
helloWorld += "rld";
```

String Methods

The String class contains a number of extremely useful methods, only some of which will be covered here. [Here is the complete API for Strings in Java 10.](#)

Importing Classes

Writing Classes

TODO Static fields

Inheritance (Subclasses)

Interfaces

Git and Version Control

Git is a Version Control System (VCS) which can be used to backup and manage large projects with numerous contributors

A repository, or repo, is the collection of folders and files which are tracked as part of a project

A commit is a change, or collection of changes, that have been made to the repository. By looking at the commit history of a project, developers can learn what changes were made when, by who, and why

Commits can also be easily reverted if the changes are later found to cause bugs, or have other unintended side effects

Git 2

A branch is a separate version of the same codebase. For example typically projects have a 'master' branch alongside various branches for different features that are in development.

The 'master' branch should contain only fully functional code and features. This may also be considered a 'production' branch

'Development' branches are used to track changes while a specific feature is being worked on. Code in these branches may not always be completely functional.

A 'merge' is the process by which changes made in one branch are combined with those of another branch

Setting up Git

To set up a new Git project you will first need to install it [here](#) (The installer's default settings are fine)

After installing Git, you should be able to right click and select the option "Git Bash Here" (pictured to the right). This will open a Command Line Interface (CLI) through which you can run git commands
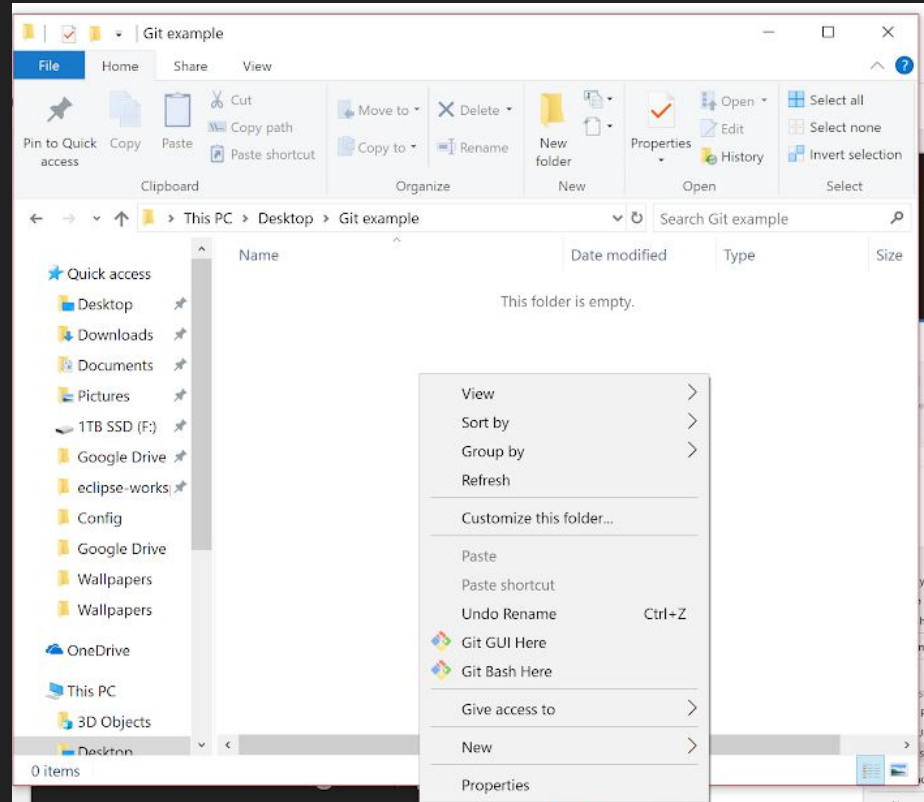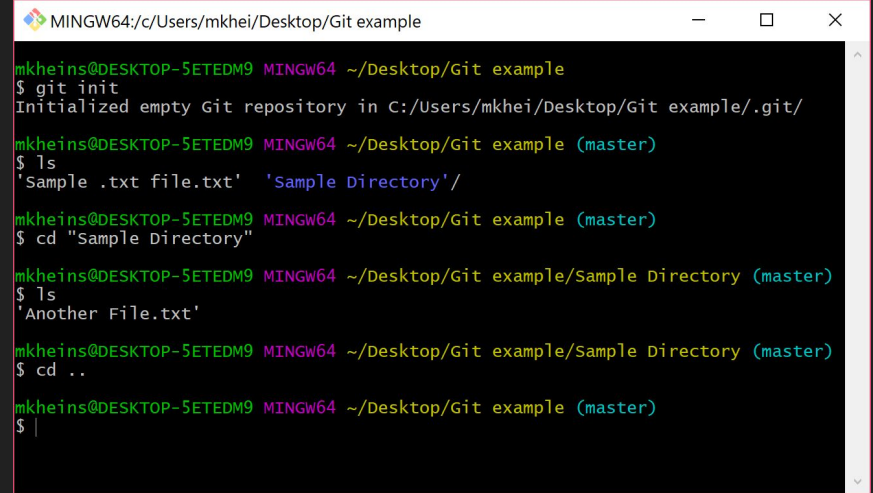
Useful Git Bash Commands

'git init' will initialize a new repository in the current directory (shown in yellow)

Every new project will have a single branch, 'master' (shown in blue)

'Cd','ls', and '..' are used to change directory, display what files/folders are in the current directory, and refer to the parent directory of the current directory

Useful Git Bash Commands 2

'git clone <repo>' will copy the repository located at the https or ssh location specified by <repo> onto the local machine, creating a subdirectory of the current directory

'git config user.name <username>' will set the name that is recorded for all commits made on the local machine. This command can be rerun at any time to change the username
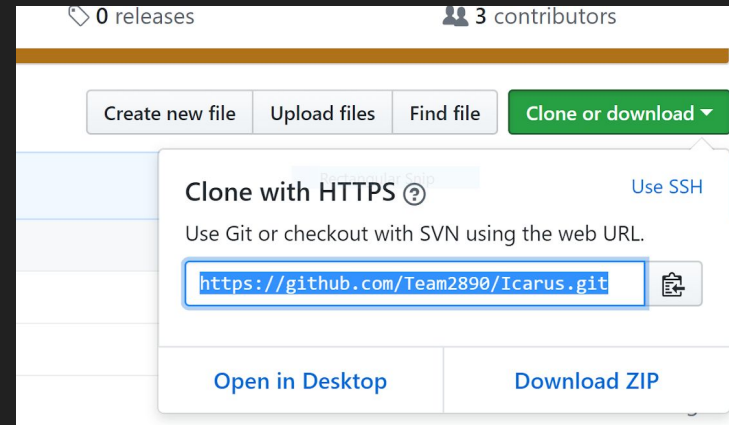
Useful Git Bash Commands 3

'git add .' will stage all changes in the current directory, as well as all subdirectories. This should be done prior to running 'git commit'

'git commit -m <message>' will commit the currently staged changes, with the commit message of <message>. The message should explain all changes made by the commit

Documentation

Useful Resources

FIRST Programming Resources Page - Instructions on how to setup your environment, along with various guides on how to program the robot using Java

Java 8 API - Complete documentation for the standard Java library

StackOverFlow - Q&A website used for crowdsourced troubleshooting, if you have a problem you can't figure out 80% chance it's on here somewhere

Team2890 Github Page - Our team's code from past seasons is kept here, along with documentation and record of historical changes.

Google - Google is your friend

Useful Resources

[Git Handbook](#) - Additional reading material on Git