Java Workshop

Goals

____ Step 1: Research what IDE to install and complete the prelab by installing and configuring an IDE

____ Step 2: Experience using an IDE and tools it provides like the debugger, compilation, and Git integration

____ Step 3: Become familiar with the general Java vocabulary and idiomatic practices

____ Step 4: Experience implementing Java software starting with basic concepts like:

A. I/O
B. Primitive types
C. Functionality of main
D. Instantiating class objects
E. Inheritance
F. Recursion

____ Step 5: Once everything is completed, move on to the optional advanced topics of:

A. Using Java collections and iterators and creating your own generic types
B. Useful usage of static and initialization blocks
C. Interfaces versus Abstract Classes and psuedo-multiple inheritance

IMPORTANT

If possible bring your own laptop to these labs so you can learn to install and work with an IDE. A couple of popular IDEs for Java development are IntelliJ and Eclipse. Use IntelliJ if you want IDE help from the TCSS. Some of IntelliJ's qualities include its robust ecosystem, intuitive interface, and reliability.

Background One — The Fundamentals

What is Java?

Java is a platform independant object oriented programming language developed by Sun Microsystems. It is comonly used for client-server web applications and is one of the most widely used programming languages. The primary appeal of using Java is that a compiled Java program works on any computer architecture that runs a Java Virtual Machine (JVM). This idea of portability through limited dependencies is the primary ethos of the language but other core values include security, simplicity, and performance.

Java is more comletely described as the Java Development Kit (JDK), a software platform made up of various components such as the compiler, debugger, and Java Runtime Environment (JRE). The JRE is a piece of software that contains everything needed to run a compiled Java program (called a Java Bytecode - similar to a C++ binary) including the Java libraries, the Java Virtual Machine (JVM), and deployment tools. The JVM is a virtual CPU that can be installed on a multitude of platforms. This is how Java achieves platform agnosticism.

Another benefit of the Java runtime is that it takes care of memory management through a process called garbage collection (GC). This provides the developer the oppurtunity to focus on other aspects of program design and can prevent issues like memory leaks and memory based security exploits. GC comes at the cost of memory consumption and inconsistent performance but for many applications these costs are negligible.

There are some drawbacks of large runtimes like the JRE. The most noteworthy being inconsistent performance. It is difficult to know when the GC will run and if you are developing performance critical code, think airplanes or stoplights, then Java might not be the tool for you. Java uses an incremental GC system so these inconsistencies are often negligible. Another issue is long start up time. The runtime is so large it takes a minute to get going. This issue has been quite the challenge for developers at Oracle to address. When you start using IntelliJ, you will understand this issue and avoid closing the IDE at all costs. Lastly, if there is a bug in the runtime there is very little you can do to fix your software. Although, given the incredible complexity of modern compilers, similar issues exist within languages that don't provide runtimes.

Why use an IDE now?

IDE's are incredibly useful for developing large projects by providing tools like graphical debuggers, program frameworks, continuous integration systems, and static or dynamic analysis. It is important to start getting used to different development environments that you may encounter throughout your education and careers. Check out the nearly 4000 plugins available for IntelliJ from their plugin repository and you will likely find something that will contribute to your workflow. One plugin that is particulary useful is the Vim plugin, which updates the key bindings and provides the same multi-mode interface. It even allows you to source a vimrc for further customization.

Vocabulary

Java developers use some different vocabulary that is worth being aware of:

- Member: A Method or a Field of a class.
- Methods: Are functions of a class. If you use the words "member function" to a Java developer they might laugh at you.
- Fields: A piece of data in a class (like an int or a String). These are sometimes called variables as well but a variable also refers to local data where a field refers to a class member.

- Referance: Its a fancy pointer. It is very differen from a C++ referance and much closer to a C++ pointer. (a C++ referance is immutable and CANNOT be null)
- final: A Java keyword that designates something cannot be changed. A final class cannot be derived from, a final variable is like a C++ const variable, and a final method cannot be overridden.
- static: A Java keyword that is the same as C++ but is worth noting. A static field is shared by all instances of the class and a static method is called using the class itself, not an instance of the class. Static methods can be invoked even when no objects of that class have been instantiated and can only operate on static fields. Static methods and variables are also often called class methods and class variables.
- super: A Java keyword to access members of a class's parent.
- Supertype (of type A): Any class or interface above A in the inheritence tree.
- this: A Java keyword to represent the instance of the class in which it appears. Although Java does not require, it is idiomatic in Java to use this.member whenever you are accessing members within a class.
- extends: A Java keyword to declare that a class is a subclass of another.
- interface: A Java keyword used to define a collection of methods and constant values. Simlar to an abstract base class with some subtle differences. Interfaces can be used as types and are heavily utilized in the Java Collections.

Tips for Success

Use Oracle's documentation. The Java version at time of writing is Java SE 13 with the most recent LTS version being Java 8. You probably downloaded the the SE 13 version of the JDK. The API documentation for SE 13 is found here:

https://docs.oracle.com/en/java/javase/13/docs/api/index.html

Oracle's documentation is incredibly detailed and organized. Any question you have about the language can be answered by reading them. In the process of looking for the answer you will learn things about the language and develop skills that are not limited to Java. The tutorials page in the documentation is also full of easy to read and detailed explanations of various topics. Stack Overflow and other tutorial websites can be useful to gather information but be careful to assess the quality. Try using these resources to find where in the documentation to look instead of massaging other's code examples to fit your application.

In extension to the previous piece of advice, consider using the Java collections. Program 5 details that you have to construct some of your own data structures from scratch but you can use the libraries for anything else you would like. I suggest looking at the ArrayList and HashMap classes.

Using IntelliJ Idea

Using an IDE like IntelliJ is helptul once you have already established strong programming skills; at that point it can be used as a tool for organizing and working on larger projects. It provides suggestions that can be helpful when you aren't familiar with the language or the project. However, these hints are only valuable if you know what the IDE is suggesting and can decide if it is actually what you want to do. IDEs can also take care of "boiler plate" code by importing templates, snippets, and project structures.
Most IDEs have community maintained plugin repositories giving you the ability to extend its functionality. IntelliJ has a particulary large community and ecosystem. It is important to note that Jetbrains products are free to students with an academic email while in school. So if you would like the ultimate version and then that is available. Although, community version does everything you need and is free to everyone.

Installing IntelliJ Idea

IntelliJ basics

Creating a Project Running and Debugging Adding Classes

Prelab One

____ Step 1: Read the background material for the lab. Follow the instructions found there to install IntelliJ idea and get familiar with the IDE. Do some research and get familiar with the enviroment so you can spend your lab time learning Java instead of installing and figuring out the IDE.

____ Step 2: java questions from old prelab

Worksheet One — First Steps in Java

In this worksheet you will go through creating classes, basic input and output, use of primitive types, the string/array/vector class, and control flow.

Introductory

• First Steps in the Java Development Environment

____ Step 1: Open up the IDE and lookup "Create a Class". Name this class Demo1 for your first Java program.

A. How do you create a class?

____ Step 2: Now create a main method to provide an entry point into your program. The syntax to do so is:

```
public class Demo1 {

   /* Send out a welcome message */
   public static void main(String[] args) {

      //comments in Java are the same as C++
      System.out.println("Welcome to CS202's Java Workshop \n" + "Enjoy!");
   }
}
```

Whoa! *Public static void main(String[] args)* let's break this down a little.

A. Public means the this method is accessable from outside the class.

B. static means the method is owned by the class and not an instance (object). This allows main to be called without ever instantiating an object of the encapsulating class.

C. main returns void now instead of int. The System object has an "exit status" field that is returned by the Java Virtual Machine when the program terminates.

D. The array of Strings as an argument is optional but is recomended. This array will contain the arguments provided on the command line (space delineated). Unlike C and C++, the first item in this array is NOT the name of the program.

____ Step 3: The System object provides an interface to access the output and error streams, the current time, and other useful tools like array copy or exit(). Take a look at Oracle's documentation of this object and see what is available.

____ Step 4: 'out' is a field of the System object of type PrintStream. System.out.println(to_print) does exactly what you would expect. Changing println to just print prints without a following new line. PrintStream also provides methods to flush the buffer and check for errors.

____ Step 5: Compile and run the program in the console window.

A. Shift F10 or Run — Run Main

B. Or, use the green "run" arrow from the toolbar by debug

• Creating Classes and Methods

____ Step 1: With the exception of a couple differences, loops, conditionals, and primitive data types are the same as C++ with the addition of some syntactical options if you prefer. Some key differences to keep in mind:

A. boolean is spelled out now

B. if statements must evaluate to Truth or False, unlike C++ where zero/NULL would be false and non-zero would be true

C. null is now lowercase

____ Step 2: Write the following method in the Demo1 class:
The method will take three integer arguments. The method will display all the numbers, each on their own line, that are divisible by the third argument and inbetween the first two arguments. Remember that you need the scope specifier at the start of every method. Write the function signiture below:

____ Step 3: Test this class by calling it from main. Make sure it works as expected! Next we will add I/O utility so the user can interact with this method.

• Creating a Utility Base Class for Generalized I/O

____ Step 1: Next we will create a utility base class that any class that needs user input can inherit from.

A. Create a new class called Utility.

B. There are many methods of working with input in Java. We will use a Scanner object. Scanners can be attached to an external data file but Java provides a FileReader class that is worth looking at.

C. At the top of this new file for the Utility class import the Scanner utility:

```
import java.io.*
import java.util.Scanner
```

D. Give the Utility class a field for a Scanner referance and write a constructor for the class that initializes it:

```
public class Utility {
   protected Scanner input;
   public Utility() {
      input = new Scanner(System.in);
}
```

____ Step 2: Let's move main() outside of the Demo1 class to clean things up a little bit. Do this by creating another class called Main and move the main function to this class. You will have to edit the entry point of your program to your new main() to run your program. (Click Run on the taskbar dropdowns and click Edit Configurations. There you will see where you can change the main class)

Intermediate

• String and Scanner Class

____ Step 1: First take a look at the Scanner and String classes. Go to the docs to see what Scanner methods are available. Some scanner methods that will be useful include:

A. input.nextInt(); //returns the next whole number

B. input.nextFloat(); //returns the next float

C. input.next(); //returns the nest single word as a String

D. input.nextLine(); //returns the next phrase ending in a newline as a String

____ Step 2: Like in C++, be aware of newlines in the input stream. Use input.nextLine(); to clear a line from the buffer.

____ Step 3: While reading in some data from the user let's look at some of the functionality of the String class. The String class has many useful methods. Go to the docs and search String. Navigate to the methods overview section and use it to fill in the return and argument types for the following methods (some may have no arguments):

A. _____equals(_____);

B. _____equalsIgnoreCase(_____);

C. _____compareTo(_____);

D. _____compareToIgnoreCase(_____);

E. _____ = String + String;
Why might you want to use the concat() method instead of the overloaded + operator?
(Hint: think efficiency)

_____

F. _____length(_____);

G. _____contains(_____);

H. _____toCharArray(_____);

I. _____toLowerCase(_____);

J. _____clone(_____);

Advanced

• Classes, Class Fields, Arrays of Objects

____ Step 1: Create a new class called Person. Give Person private String fields for first/last name, float fields for hourly_wage/hours_worked, and int fields for age/dollars/quarters/dimes/nickles/pennies. Write a constructor for Person to set these fields to some defaults. (null for the Strings, 0 for ints and floats)

```
public class Person extends Utility {
   private String first;
   private String last;
   private float hourly_wage;
   private float hours_worked;
   private int age;
   private int quarters;
   ...

   public Person() {
      first = null;
      last = null;
      ...
   }
}
```

____ Step 2: Give the person class a public build() method. Have this method prompt the user to set the values for all the fields. Remember, the scanner methods return the values.

```
this.first = this.input.next();
```

The 'this' keyword isn't required in this context but it is idiomatic Java to use it whenever accessing fields of the current class.

____ Step 3: Test the build method by creating a person from main() and calling the build function. (from the main class)

```
public class Main {
   public static void main(String[] args) {
```

```
   }
```

____ Step 4: Now add the following methods to the Person class:

A. display() — displays a Person's details

B. work(float hours) — adds an ammount to a Person's hours_ worked field

C. pay() — uses the hours_worked and hourly_wage fields and adds to the dollars and coins fields.

____ Step 5: Add an array of Strings for performance reports. Set it to null in the constructor. Show how to add this field to the class:

And show how to create an array of 5 strings called reports:

And add a method that adds a performance report for the person by asking the user to input a report. Make sure to take care of the special cases! Here are some tips:

A. The array object has a length field (array.length). This is the CAPACITY of the array, not the number of items in the array. You have to keep track of that yourself with another variable OR iterate through it searching for null referances.

B. You can copy the content of an array using a for loop OR using the System method:
System.arraycopy(Object src, int srcPos, Object dest, int destPos, int length);
See the docs for more details.

____ Step 6: Last, create a method in Person to help you sort people by their last name. The function should take another person as an argument and return an int. Use the String method compareToIgnoreCase(src) to determine which person comes first.
REMEMBER: Java overloaded the + operator for the String class but not the $==$, $>, <, \leq$, or $\geq$ operators. So do not try to check string equality with $==$. Doing so will compare the referances to the actual Strings themselves.

Background Two — Recursion and Beyond

Part two of the Java workshop is mostly practice with recursion in java. Recursion in Java is quite different than recursion in C++ because Java doesn't have pass by referance. Some things to keep in mind are:

A. Every non-primitive type and arrays of primitives types are Java Referances. The referances are most similar to C++ pointers.

B. You can only pass these referances by value. That means mutating an argument inside of a function only mutates it inside of the scope of that function. It remains unchanged in the calling routine.

C. Remember that nice recursion trick from C++ where you passed pointers by referance? Now there isn't a nice way to do that.

There are a couple of different ways to handle this issue. One possibility is to create a class that encapsulates the referance. This way, when you mutate the referance, it is changed within the class. This solution is not ideal, as it requires superfluous abstractions and methods. Another option would be to insert the referance into an array of length one. This is similar to creating a pointer to a pointer in C++. This trick may be usefull occasionally but is honestly quite kludgy and unnatural. When others read your code they will be confused why your method takes an array when it only makes sense to pass in a single instance of the argument type.
The recomended solution would be to return the modified value back to the calling routine. This way everything gets linked up correctly on the backtrace through the calling stack.
Here is an example of how you could implement remove with this method for a LLL of integers:

```
public void remove(int to_rm) throws EmptyList, NotFound
{
   if(head == null)
      throw new EmptyList;
   this.head = remove(this.head, to_rm);
}

protected Node remove(Node head, int to_rm) throws NotFound
{
   if(head == null)
      throw new NotFound(to_rm);
   if(head.get_data() == to_rm)
      return head.get_next();
   head.set_next(remove(head.get_next(), to_rm));
   return head;
}
```

abstract base class vs interface using collecions and std static, initialization blocks
unnamed functions? closures / lambdas?
debugger basics?

Prelab Two

The prelab for the second part of this workshop is to complete a fully recursive linked list implementation. This will be the class you work with to complete this lab. The first half of this lab focuses on recursive linked list manipulations and the second advanced portion highlights some more advanced Java topics.

____ Step 1: Here is the C++ code to remove all evens recursively, utilizing pass by referance, returning the new length of the list. (Note: the Node's get_next() method returns a referance to a pointer)

```
int remove_even()
{
   if(!head)
      return 0;
   return remove_even(head);
}

int remove_even(node* & head)
{
   if(!head) return 0;

   if(head.get_data() % 2)
      return 1 + remove_even(head.get_next());

   node* to_rm = head;
   head = head.get_next()
   delete to_rm;
   return remove_even(head);
}
```

Modify this C++ to use pass by value by returning what will be head's next.

____ Step 2: Now convert this to Java code:

____ Step 3: Write a recursive build function in Java that gets each data from the user, asking the user each time if they would like to add another item to the list.

____ Step 4: Create a Linked List class in Java that fufils these requirements:

A. If you are using a Node class, there shouldn't be any public class methods that take a Node as an argument or return a Node.

B. The data type of the list will be an integer.

C. Tail referance is optional.

D. Methods should include:

- insert - takes an int as an argument
- remove - takes an int as an argument
- build - from Step 3 of this prelab
- display - no arguments
- clear - empties the list
- sum - returns the sum of the list
- avg - returns the average as a float

E. All of these methods should be implemented recursively. No loops allowed.

Worksheet Two

Introductory

____ Step 1: Count the number of times the last item appears in the list.

A. Recursive prototype:

_____count_last(_____);

B. What is the base case?

C. What work should be done in the wrapper?

D. What work should be done before recursive call? (if any)

E. What work should be done after the recursive call? (if any)

____ Step 2: Remove all the odd numbers from the list.

A. Recursive prototype:

_____remove_odd(_____);

B. What is the base case?

C. What work should be done in the wrapper?

D. What work should be done before recursive call? (if any)

E. What work should be done after the recursive call? (if any)

Intermediate

____ Step 1: Copy the list. The wrapper should take another list class as an argument.

A. Recursive prototype:

_____copy(_____);

B. What is the base case?

C. What work should be done in the wrapper?

D. What work should be done before recursive call? (if any)

E. What work should be done after the recursive call? (if any)

____ Step 2: Remove all but the last two items of a linked list.

A. Recursive prototype:

_____remove_except(_____);

B. What is the base case?

C. What work should be done in the wrapper?

D. What work should be done before recursive call? (if any)

E. What work should be done after the recursive call? (if any)

____ Step 3: Remove the smallest and largest items of a linked list.

A. Recursive prototype:

_____remove_big_small(_____);

B. What is the base case?

C. What work should be done in the wrapper?

D. What work should be done before recursive call? (if any)

E. What work should be done after the recursive call? (if any)

____ Step 4: If the sum of the list is even, copy all the even data. If it is odd, copy all the odd data. Do this in a single traversal.

A. Recursive prototype:

_____copy_sum_condition(_____);

B. What is the base case?

C. What work should be done in the wrapper?

D. What work should be done before recursive call? (if any)

E. What work should be done after the recursive call? (if any)

Advanced

The advanced section of this lab covers some basic information on collections, interfaces, and initialization blocks. These are not required for your programs or exam but could potentially help out with your Java programming assignment.

• Collections

• Interfaces and Multiple Inheritance

• Initialization Blocks

____ Step 1: placeholder

arraylist collection multiple inheritence using interfaces static, initialization blocks