/* **Scenario:** I am working for a multinational bank that would like to develop a web application. The company has worldwide distribution, and as such their online application must work across platforms. The company would like the web app to be developed from scratch; there is no legacy code. The company plans to make changes to the product in the future by adding and modifying features. The team I work for is ten programmers distributed around the world, ranging in programming ability. No language has yet been decided for use in the project, and the team has proposed we each submit a paper advocating for a particular language. The audience is the entire team rather than the team leader. */

I. Introduction

Given the nature of our project and team, I have chosen to advocate for the use of Java as our choice of programming language. The Java philosophy and design combined with the efficiency and security the language provides means the language is the clear choice for our project. It not only offers short-term benefits for our team as we develop the code, but sets our company up for success as our product grows long-term. Below I have gone into detail about the above points, providing some examples for clarification. If taken into consideration, the use of Java will positively impact our team's workflow and final output.

II. Java Philosophies

Java is highly portable. Our company's client base has global scope, and as such, our web application must work on a variety of machines and systems around the world. Not only this, but the remote nature of our team means that code we write must be transferable and testable across systems. Java's core philosophy is to "write once, run anywhere" (Louden and Lambert,

162-163). Java code compiles to bytecode independent of the architecture of its local machine (Louden and Lambert 162-163). The byte code is then interpreted by the Java Virtual Machine (Louden and Lambert 162-163). This philosophy of software development is ideally suited for web-based projects, such as ours, because it frees programs from machine-specific requirements and restrictions (Louden and Lambert 162-163).

Along with its portability, Java's object oriented design would benefit our workflow. The project we are to develop is directly related to real-world situations and concepts. With this in mind, it is imperative the language we choose not only allows for the conceptualization of real-world phenomena, but holds this as its core philosophy. Java is a purely object oriented programming language (with the exception of primitive types) (Louden and Lambert 163). This means, for example, we can structure our code as a series of interacting objects, much like how our company's customers interact when they are on location (Louden and Lambert 163). For example, we may design customer and currency classes to hold client information and model the processing of transactions.

Not only is Java object oriented, but it is also impure, particularly in terms of user interaction. Java provides a variety of classes and methods that make accessible the keyboard and mouse. For example, the `MouseListener` interface and `Scanner` classes are commonly used to accept and convey information to the user ("How to Write a Mouse Listener" and "Scanner"). In the case of the Scanner class for user input from the keyboard, consider the following (assuming the user will input the correct type):

```
Scanner reader = new Scanner(System.in);
System.out.println("How much would you like to withdrawal?");
float withdrawalAmt = reader.nextFloat();
reader.close();
```

From the above, we have obtained the amount of money a particular client would like to withdrawal. With an application as interactive as ours, it is paramount that our language of choice is one that facilitates user interaction. Java holds user interaction as part of its design, making it relatively simple to implement an interactive design.

III. Efficiency

Along with its philosophies, Java provides efficiencies from which our team would greatly benefit. Java is a language with extensive libraries to support applications with a range of goals and functionalities (Louden and Lambert 162-163). Among Java's libraries include data structures. This is especially useful in our team's case; we do not have any legacy code around which to structure our design. In section I paragraph II, I mentioned the potential use of classes to structure client transactions. Consider the following example:

```
import java.util.*;
...
Hashtable<String,String> loginCredentials =
             new Hashtable<String,String>();
... (loginCredentials is added to as users interact with the
program)

Scanner reader = new Scanner(System.in);

System.out.println("Please input your username:");
String username = reader.nextLine();
System.out.println("Please input your password:");
String password = reader.nextLine();
reader.close();

if(loginCredentials.contains(username)) {
     String maybePassword = loginCredentials.get(username);
     if(maybePassword.equals(password)) {
          //log in to the system
     }
}
```

The above example shows the use of `Hashtable` from `java.util.*` to check if a user's username and password match the data stored in the system. Writing our own Hashtable class would be unnecessary. The ability to import data structures as demonstrated above saves time, allowing our team to focus on more important project tasks.

In addition to Java's large set of libraries, Java's compile-time type checking provides efficiencies for our team. Java is a strongly static typed language (aside from polymorphism and other paradigms) (Alomari et al. 17). This allows programmers to determine errors at compile time, rather than during execution as in dynamically-typed languages (Alomari et al. 17). Java's static type system also means optimizations may be performed at compile type (resulting in more efficient execution) and that debugging tools are better able to determine functionality of a particular section of code (Alomari et al. 17). In this vein, Java's types make code more readable. Consider the following example in Python, an untyped language, in which some text file was sent from one branch in our company to another, then processed:

```
def internationalTransferMemo(textFile):
    allSent = nltk.sent_tokenize(textFile)
    numSent = len(allSent)
```

The above example poses some challenges, and as a result, is not very readable. Mainly, what are the types of `textFile`, `allSent`, and `numSent`? Figuring out the types of this code, and further, what this code does, would require searching through the Python specification. Moreover, there are no protections against assumptions the code makes about the type of `textFile`; passing an integer value to `internationalTransferMemo` would still compile, and lead to errors during execution. In Java, there is no such ambiguity. This is

particularly advantageous for our team, since we do not have the luxury of writing and explaining code face-to-face, or of searching for errors caused by erroneous typing.

Keeping in mind the Java type system, Java provides long-term efficiency in terms of refactoring. As the company grows and would like to add new features to their e-commerce, or modify existing features, our team will likely need to refactor. Java's type system makes this process more efficient in comparison to an untyped language, as a result of compile-time type errors. For example, consider some refactoring scenario in which a type of a particular component must be changed. A programmer may simply follow the compiler errors to fully update the code, rather than search through the entire source code. This is especially important to bear in mind in a project as large as ours.

IV. Security

Though efficiency is important in the workflow of our team, it is also important to keep in mind the core values of the company for which we are creating this project. As a multinational bank, protecting the information and assets of our clients is key to our company's success. It follows that the web application our team develops adheres to these values. Java provides tools to make our project semantically safe, and to encapsulate vital sections of code. The garbage collector is an example of semantic safety in Java (Alomari et al. 17). The garbage collector handles the allocation and deallocation of memory, preventing memory leaks (Alomari et al. 17). This ensures the integrity of the program, and that important memory holding client information is not erroneously deallocated, or unnecessarily left behind. Consider the following code in C, a language in which garbage collection must be handled by the programmer:

```
void clientData(double socialSecurityNum){
    double *p;
```

```
    p = (double*) malloc(sizeof(double));
    *p = socialSecurityNum;
}
```

In the above, the pointer `p` is not accessible outside of `clientData`, and has been leaked. Memory leaks pose potential security threats. Hackers wishing to exploit our company's system may be able to perform a denial of service attack by crashing the program, or trigger other unexpected program behavior caused by exhausting memory ("Memory Leak"). The garbage collector ensures that such a situation would not occur by protecting against said memory leaks.

Along with the garbage collector, Java modifiers make its programs more secure. Java access modifiers provide encapsulation (Alomari et al. 16). In particular, `private` and `protected` modifiers restrict accessibility to classes, methods, and attributes as they interact with other, potentially damaging program components (Alomari et al. 16). In combination with this, the `final` keyword ensures variables will maintain their values, methods cannot be overridden, and classes cannot be subtyped. Hackers may subvert a particular system by adding subclasses of a class and replacing the original class with their malicious code ("Writing Final Classes…"). Through this method, attackers may gain private information or further attack the system ("Writing Final Classes…"). Java modifiers, particularly by declaring a class `final`, protect programs from such an interaction with foreign code. This is essential when handling our clients' assets.

V. Conclusions

I believe Java is the best selection for our project. It's core philosophies align with our internationally-based team and with the interactive, real-world goals for our web application. Use of Java's libraries results in efficiencies in writability, and Java's type system makes its

programs unambiguous and readable, which is necessary for the remote nature of our team. Further, its compile-time type checking allows Java programmers to more efficiently refactor code, as will be the case as our company expands its online presence. Additionally, Java provides securities that can be harnessed to combat attacks. The garbage collector and Java modifiers protect against the exploitation of our clients' data. Selecting Java as our programming language would greatly improve our team's workflow now and in the future, resulting in a more robust product for our company.

VI. Sources

https://www.researchgate.net/publication/274572185_Comparative_Studies_of_Six_Programming_Languages

https://docs.oracle.com/javase/7/docs/api/java/util/Scanner.html

https://docs.oracle.com/javase/tutorial/uiswing/events/mouselistener.html

https://www.owasp.org/index.php/Memory_leak

http://journals.ecs.soton.ac.uk/java/tutorial/java/javaOO/final.html