Assignment 5

Nowadays, both java and python are one of the most popular programming languages in use. Here are the 8 primary constructs with the design choices taken in java and python.

The first primary construct is scope. Sebesta [1] says that scope of a variable is the range of statements in which the variable can be referenced or assigned. In the case of Java, Liang [2] shows that the scope of a local variable starts from its declaration and continues to the end of the block that contains the variable. Every control statement, class and function create their own scopes. In Python, Maurer [3] points that variables are scoped to the innermost function, class, or module in which they're assigned. As a result, Java cannot read variables declared in an if block while Python can still reference variables in control statement structure. Java's decision improves readability as variables' scope are clearly marked, while python reduces cost and improves writability as simple scoping design makes the language easier and takes less time to learn.

```java
//Java//
public class Scope {
    public void main(String args[]){
        String name = "EXP";
        if (name.equals("EXP")) {
            boolean found = true;
        }
        if(found) {
            System.out.println("Found EXP!");
        }
    }
}
```

```python
//Python//
name = "EXP"
if name == "EXP":
    EXP_found = True
if EXP_found:
    print "I found EXP!"
```

The second primary construct is array. Sebesta [1] explains that an array is a homogeneous aggregate of data elements in which an individual element is identified by its position in the aggregate, relative to the first element. In Java, Sebesta [1] pointed that once Java initializes an array, there is no way to expand such an array. Attempts to add an element will cause an out of bound error. However, Python arrays are resizable, making array modification possible. The position of element changes as an element in the middle is deleted. The design choice of Python thus has better writability as it reduces the hassle of referencing array out of bound. The design choice of java has better reliability as the index of an element will not change if the element in front of it is deleted, avoiding accident reference to wrong element in an array.

```java
//Java//
String [] names = {"Ben", "John", "Charles"};
names[3] = "Bill"; //exception
```

```python
//Python//
names = ["Ben", "John", "Charles"]
names.append("Bill")
```

The next primary construct is selection statement. According to Sebesta [1], Both python and java have two-way to multiple-way selection statement. Instead of using keywords such as "then" or "end" in the control expression, Java, in the case of "if…else…" statement, gives the user an option of using brackets to distinguish the statements. Python, on the other hand, uses indention to distinguish the then and else clauses. In the case of "if…else…if", python uses "elif" keyword to specify the extra condition the selection statement needs to check. Java's design choice is better than Python as it has better readability and reduces cost: the brackets in place will be so much clearer than indention, and the use of existing English word as keyword instead of creating a fusion word makes learning the language easier.

```java
//Java//
public class Scope {
    public void main(String args[]){
        String name = "EXP";
        if (name.equals("EXP")) {
            System.out.println("Found EXP!");
        }
        else {
            System.out.println("Nothing Found!");
        }
    }
}
```

```python
//Python//
name = "EXP"
if name == "EXP":
    EXP_found = True
if EXP_found:
    print ("I found EXP!")
else:
    print ("Nothing Found!")
```

The next primary construct is iterative statements. Both Java and Python has iterative counter-controlled statements, both with keyword of "for". Java, similar to C, creates a different scope for every variable in the loop. Immediately after the keyword follows the statement for counter setup, a statement for criteria, and a statement for increment. Each statement is separated by semicolon. When multiple statement in the loop body exist, Liang [2] recommends brackets for identification. In Python's case, the counter variable is immediately after the keyword, and keyword "in" and a function is used to specify starting value, ending value, and increment. The statement after is distinguished using indention. Java has better readability and reliability because it specifies each expression clearly, and thus reduce the possibility of mistakes. Python has better writability, as loop creation is so much simpler and quicker than Java.

```java
//Java//
public class Scope {
    public void main(String args[]){
        for(int i = 0; i < 10; i++){
                System.out.println(i);
        }
    }
}
```

```python
//Python//
for i in range(0, 10, 1):
  print(i)
```

The next primary construct is the parameter passing method of subprogram. Sebesta [1] summarizes three modes of parameter passing: in mode, out mode, and inout mode. Java only allows in mode, the pass by value mode when calling a subprogram, whereas Python can either be configured as pass by value (in mode) or in out mode. As a result, any modification to the parameter will not change after subprogram execution in Java, but according to the different implementation, the parameter passed in could cause difference. In conclusion, Java has better reliability and writability, because it is harder for functions to accidentally change the value of original function, protecting the integrity of data, and the writing of code easier for function. Python will have better writability if the intention of subprogram is to swap two values, freeing the difficulties risen by objects.

```java
//Java//
public class Scope {
    public void main(String args[]){
        int a = 10, b = 5;
        sub(a,b)
        System.out.println(a);
        System.out.println(b);
    }
}

public int sub(int a, int b){
    int temp = a;
    a = b;
    b = temp;
}
```

```python
//Python - inout//
def sub():
        temp = a
        a = b
        b = temp
a = 10
b = 5
sub()

//Python - in//
def sub(a,b)
        temp = a
        a = b
        b = temp
a = 10
b = 5
sub(a,b)
```

The next primary construct is overloaded subprogram. In Java, Liang [2] points that subprograms can be overloaded in class as long as the arguments or the return type is different. When the function is called, Java checks the input type, and automatically matches the function with the correct input combination. Python can also perform overloading on subprogram, but it has to setup a series of if else statements to help identify the correct procedure of the function. As previously described, Java is better in writability and reduces cost, due to the effort of using the same function to do different things is lesser, and the training of new programmer in this aspect is simpler. Python is better in readability, as the selection process of which function will be used can be simply traced.

```
//Java//
public class Scope {
    public void main(String args[]){
        int a = 10, b = 5;
        sub(a,b)
        System.out.println(a);
        System.out.println(b);
    }
}

public int sub(int a, int b){
    int temp = a;
    a = b;
    b = temp;
}
public int sub(double a){
    int temp = a;
    a = 21;
}
```

```
//Python//
def sub(a, b=None):
        if b is not None:
                temp = a
                a = b
                b = temp
        else:
                temp = a
                a = 21


a = 10
b = 5
sub()
```

The next primary construct is overloaded operator. In Python, Sebesta [1] reveals if there are two user-defined class objects adding each other, the user can define the detail operation in using special “def” function. In Java, operator overloading is strictly restricted to the pre-defined string concatenation and other logical operation. As a result, Java retains better orthogonality and reduces cost, as codes are simpler to understand, and programmers does not need to remember the meaning of symbol when they go form one program to another. Python has better writability as any operator can be reconfigured to accomplish complex operation using simple operator.

```
//Java//
System.out.println("first string" + "second string");

//Python//
def __add__ (self, second):
return Complex(self.real + second.real, self.imag + second.imag)
```

The last primary construct is Data abstraction. Sebesta [1] believes an abstract data type is an enclosure that includes only the data representation of one specific data type and the subprograms that provide the operations for that type. The process of providing only the essentials and hiding the details is known as abstraction. [4] The “class” provided in both Java and Python is the abstraction of data. Classes provide a means of bundling data and functionality together. Creating a new class creates a new type of object, allowing new instances of that type to be made. [5] Creating and calling class is easier in python, boosting writability and cost. Java does better in readability as attributes and methods are clearly distinguishable.

```
//Python//
class Complex:
    def __init__(self, realpart, imagpart):
         self.r = realpart
         self.i = imagpart

x = Complex(3.0, -4.5)

//Java//
class Complex{
        double real;
        double imag;
        Complex(double new_realpart, double new_imagpart){
                real = new_realpart;
                imag = new_imagpart;
        }
}

public class Comp{
    public void main(String args[]){
        Complex c_num = new Complex(3.0, -4.5);
    }
}
```

References

[1] R. W. Sebesta, *Concepts of Programming Languages*, New York, NY: Pearson, 2019.

[2] Y. D. Liang, *Introduction to Java Programming Comprehensive Version*, Upper Saddle River, NJ: Pearson, 2015.

[3] L. Maurer, "What's the scope of a variable initialized in an if statement?," Stack Overflow, 2010. [Online]. Available: https://stackoverflow.com/questions/2829528/whats-the-scope-of-a-variable-initialized-in-an-if-statement. [Accessed 27 Nov. 2019].

[4] Python Software Foundation, "Python Documentation," 2019. [Online]. Available: https://docs.python.org/3/contents.html. [Accessed 27 Nov. 2019].

[5] GeeksforGeeks, "Abstract Data Types," 2019. [Online]. Available: https://www.geeksforgeeks.org/abstract-data-types/. [Accessed 27 Nov. 2019].