SAINT LOUIS UNIVERSITY
SCHOOL OF ACCOUNTANCY, MANAGEMENT,
COMPUTER AND INFORMATION STUDIES
DEPARTMENT OF INFORMATION TECHNOLOGY

IT 311

KOTLIN CODING STANDARDS

SUBMITTED BY:
Abelanes, Lourdes Jaira
Alfiler, Valjunyor
Aromin, Eddyson Tristan
Dumlao, Villamor
Sales, Rafael Miguel
Salinas, Adrian
Sanchez, Jan Joseph
—
9487

SUBMITTED TO: Mr. Roderick Makil

CODING STANDARDS

1. Naming conventions

a. Packages should be named in lowercase. For example, transactionprocedures has an ATMTransactions class:

```
package transactionprocedures.ATMTransactions
```

b. For class and Kotlin file names, use camel case naming convention. For instance, a class names can be named as:

```
MainClass.kt
ObjectClasses.kt
```

c. Naming conventions for variables should be defined as what property it should be and multi-worded properties should be separated with an underscore "_". For example, a bank slip has the following fields: account number, account name, and amount withdrawn.

```
var account_number: String
var account_name: String
var amount_withdrawn: Double
```

d. Constant and Non-Constant Names

   i. Uses all uppercase letters and the words are separated by underscores. Constants are properties with no custom function wherein the contents are deeply fixed and the functions have no detectable side-effects.

```
const val NUMBER = 20
val ID = listOf("Marc", "Robert")
```

   ii. Use Camel-case for Non-Constant Names. Non-constant can only be called by non-constant objects.

```
val variable = "var"
val nonConstScalar = "non-const"
```

e. For object variables, the name of the object should be defined as is in its plural form. For example, we have an object "Request," it can be rewritten as:

```
var requests: Request
var bank_accounts: BankAccount
```

f. Functions should be named in a camel-case format. For example, we have a getEmployees and getDepartments functions, it should be written as:

```
fun getEmployees() {
        // do something
}

fun getDepartments() {
        // do something
}
```

2. Declarations and Imports

a. All import statements should be sorted in a lexicographical manner at the top-most part of the class or first function.

b. All variable declarations should be sorted according to how the data is sorted in the database or data headers in a table. For example, an Employee table has the following headers, Employee ID, Employee Name, and Department ID. When declaring variables, it should be written, according to the order of the table.

```
fun getEmployees() {
        var employee_id : Int
        var employee_name: String
        var department_id: Int

        // do something
}
```

3. Data Classes

a. Data objects should be placed in one class. For instance, a class named DataClasses.kt may contain the following:

DataClasses.kt file:

```
data class ATMTransaction(var account_number: String, var account_name: String, var transaction_type: String, var amount: Double)

data class ATMAccount(var account_number: String, var account_name: String, var balance: Double)
```

b. In cases where data classes can be grouped together based on category, it should be grouped together and separated by Kotlin classes according to category.

4. Classes

a. Classes should always define object properties, behaviors, and functions.
b. Declare global fields in private.
c. Always declare the primary constructor as a default constructor.
d. Always declare the secondary constructor to accommondate different fields.
e. Always put a setter and a getter.

5. Use of Conditional Statements

a. The use of “when” statement

i. Use “when statement” for conditions that are specific.

Example:

```
when (program_course) {
        "BS Information Technology" -> println("$program_course  is from SAMCIS")
        "BS Electronics Engineering" -> println("$program_course is from SEA")
        "BS Medical Laboratory Science " -> println("$program_course  is from SONAHBS")
        "BA Communication" -> println("$program_course  is from STELA")
        "BS Management Accounting" -> println("$program_course is from SAMCIS")
        "BS Geodetic Engineering" -> println(“"$program_course  is from SEA")
        "BS Pharmacy" -> println("$program_course is from SONAHBS")
}
```

b. The use of “if-else” statement

i. Use “if-else ” if there are multiple conditions that need to be specified and satisfied.

Example:

```
if (samcis_course == "BSIT" || samcis_course == "BSCS") {
        println("$samcis_course is a CS/IT/MMA course.")
} else if (samcis_course == "BSTM") {
        println("$samcis_course is a HTM Course");
} else if (samcis_course == "BS Accountancy" || samcis_course == "BS Management Accounting") {
        println("$samcis_course is an ABLT Course")
} else {
        println("No course specified")
}
```

6. **Use of Functions**
    a. Functions should always be in its optimized or simplified form as possible, ideally less than 20 lines, but not more than 30 lines.
    b. If a function requires code that exceeds more than 30 lines, extract necessary blocks of code to create a helper function.
    c. Functions should only have one specified task. For example, a file named ArithmeticOperations contains functions that should add, subtract, multiply, and divide; it should only contain functions specifically for adding, subtracting, multiplying, and dividing. Thus, ending with functions named:

        add(var n1: Double, var n2: Double)
        subtract(var n1: Double, var n2: Double)
        multiply(var n1: Double, var n2: Double)
        divide(var n1: Double, var n2: Double)

    d. Group related functions together and sort them according to the most used ones.

7. **Indentations**
    a. Indent every block of code.

```
fun function() {
        if (condition){
                // do something
        } else {
                // do something
        }
}
```

    b. Using two space indentation for blocks.
    c. Use of four spaces without utilizing tabs.
    d. Usually starts at braces at the end of the line; closes the braces on a separate line that is aligned horizontally.

```
if (elements != null) { // Curly brace starts at the end of the line
   for (value in elements) {
      // do something
   }
} // Closes the block with separate braces
```

8. **Spaces**
    a. Reversed words such as if, for, catch, else and etc… from an open parenthesis, curly braces that precedes on the same line are suggested to have a single ASCII space or to be separated,which ensures clarity to the code. Example:

```
// Example 1.
for (i in 0..1) {
}

// Example 2.
} else {
}
```

b. Some Operators, and “operator-like” symbols such as lambda expressions are also suggested to apply the single ASCII spacing practice on both sides. Example:

```
val ten = 5 + 5 // Example 1.

ints.map { value -> value.toString() } // Example 2.
```

c. Operators such as (::),(.),(..), and etc… of a member reference are not suggested not to apply to the single ASCII space. Example:

```
val toString = Any::toString // Example 1.

it.toString() // Example 2.


for (i in 1..4) { // Example 3.
  print(i)
}
```

d. The colon (:) and comma (,) apply the single ASCII space only if used in a class declaration for specifying interfaces, base class, or when used in a where clause for generic constraints.

```
class Foo : Runnable // Example 1.

val oneAndTwo = listOf(1, 2) // Example 2.
```

e. Multiple spaces are allowed to both sides of the double slash (//), it is suggested, yet not required.

```
// Example 1.
var debugging = false // disabled by default
```

9. Exception Handling

a. Use a try-catch block to control an error that will occur in that block of code especially if the caught error is needed for a specific purpose such as throwing a message when an exception occurs.
b. Use the throws keyword instead of the try-catch block when there is no specific purpose of catching an error.
c. Kotlin has a nothing keyword used in handling exceptions. Use the nothing keyword to mark codes that could not be reached.

10. Code Documentation Comments

a. All variables in a class should have documentation comments. If the code only uses support variables, only one-line comments are needed.
b. All functions should have documentation comments, including the description of the parameter and return values if they exist.
c. All Kotlin files should have a multi-line comment at the top-most lines of the code that describes the set of functions being done in the file, as well as the author/s and copyright.

11. Variables and Collections

a. Use immutable variables rather than mutable variables, especially when dealing with values that should not change after initializing it.
b. For collection interfaces, prefer to use immutable collections (listOf, setOf, mapOf), assuming they do not mutate. Use mutable collections (arrayListOf, hashSetOf, mutableMapOf) for situations that require dynamic control of values.

12. Cross-platform Utilities

a. In cases where Java functions will be used, be separated in packages away from Kotlin codes.
b. Can share standard code in most all platforms that can be used in a project.

13. Other Considerations

a. Refrain from using temporary variables.
b. Refrain from combining Java with Kotlin code, although they are compatible.