Fantom

Fantom PL

CPCS301

Part: 2

Supervised By: Asif Irshad

Group Members:

	Name	ID	Roles	Description
1	**Abdulrahman Mohammed Alqarni**	2135106	**Team Leader** **Multimedia Specialist**	• Ensure deliverables are set on time • Manage and coordinate between members • Manage all media needed for this project
2	**Mahmued Ahmad Alardawi**	2135209	**Technical Writer**	• Create and design visual work for the project • Write and gather information needed for this project • Research subject experts

Table of Contents

Report 1

1. Brief Introduction of Fantom PL

Fantom is an advanced, open-source high-level general-purpose programming language designed to be as versatile and user-friendly as possible It is a software designed by Brian Frank developed in 2005 that became famous in the Java Virtual Machine (JVM) community.

Minimalism is one of the major features that characterizes Fantom. It has a simple syntax and is ideal for beginners. Furthermore, Fantom has various programming paradigms like imperative and functional object-oriented programming and it's highly modular.

Standard library supporting programs in offering necessary facilities and equipment are also provided by Fantom. It is also compatible with several platforms such as Windows, macOS, and Linux with no platform dependency.

Fantom is an intuitive programming language easily be applied in various fields.

2. Description of the suggested domains that can apply Fantom

The term "domain" refers to an area or field of application where a programming language is commonly used or that best suits a particular programming language in the context of programming languages. This symbolizes the areas of application or issue domains for which a language is intended or widely spoken. A language's features, libraries, performance, community support, and other aspects are used to decide which one is suitable for an area. In most cases, various types of specialized tools, frameworks, and libraries geared towards specific domains, are available for different languages.

1. Web development: Web development involves the use of inbuilt HTTP and RESTful web service capabilities by Fantom. Moreover, it comes equipped with several developmental tools and frameworks that are essential in creating web apps.

2. This low-level programming language Fantom suits the system programming that we are dealing with in this case. It can be used efficiently for writing different low-level software packages such as those that write operating system parts, device drivers, and so on.

3. Scientific computing: This type of computation is appropriate as several Fantom

libraries and tools are available. It allows for native support of numerical calculations, linear algebra, and other mathematical processes.

4. Game development: Fantom can work with 2D and 3D graphics as well as audio and input devices, making it suitable for game development. It also provides numerous developmental tools and game-making environments.

5. Networking: The native support of Fantom encompasses TCP/IP, UDP, and other networking protocols. It can be used as a good language in making the middle wire, server, and other network applications.

6. Using the available Fantom libraries, this programming language makes it possible to develop mobile apps that run on iOS and Android.

7. Machine learning: It is an advantage that Fantom can carry out numeric calculations and linear algebra as it is a good language for machine learning. Moreover, it has a few frameworks and tools that allow one to easily develop machine learning models.

Fanton is a versatile language which can be used in many areas. It is suitable for both, the more experienced users and those coming newly into the field.

3. Description of Fantom PL Paradigm

The reason for this is that Fantom supports multiple paradigms. It is mainly an object-oriented language that also supports imperative and functional programming.

1. Object-oriented programming is a paradigm written in terms of objects of programming. An object contains data and behavior as a whole and this object is a self-contained piece of code. The use of objects provides representation for actual entities and concepts of OOP. Support for OOP is provided by the class-based object model in Fantom that allows programmers to declare classes, objects, and methods.

2. FP is a programming paradigm that is based on functions. In this case, functions are treated just like ordinary variables in FP. Fantom supports higher-order functions, closures, and immutable data structures, making FP.

3. Imperative programming is based on statements that modify the status of a program. Instead of “what to do”, imperative programming focuses more on “how to do”. Fantom supports loops, conditionals, and others such as imperative programming.

In summation, Fantom is a powerful language that can be used for many purposes since it implements several programming paradigms. Programmers have options of OOP, FP, and imperative programming as their paradigms while in their work.

4. Evaluation Criteria Judgments for Fantom PL

Readability:
Fantom was developed to provide easy-to-read code and simple, understandable grammar. Its simple syntax helps in reducing the amount of boilerplate codes used for accomplishing tasks. It has an excellent and wide internet lesson and guidelines repository that is well documented. Overall, Fantom is quite straightforward as far as learning and usage go, while remaining highly legible.

Writability:
Fantom is an extremely writable language because it supports multiple programming paradigms including imperative, functional, and object-oriented programs. It exhibits an understandable syntax which makes it easy to write powerful coded expressions based on a wide range of built-in properties. Secondly, it has several libraries and frameworks that help in the programming of complex applications.

Reliability:
The language of Fantom is trusted as it was designed for stability and security. It comes with inherent functionality like, for instance, typing safety, memory safety, or exception handling, which simplifies the creation of trustworthy and mistake-free coding. The standard library for the language is also very well-defined; and should be both reliable and trusted.

Cost:
This language is open source and free, hence disseminating and implementing it does not incur any cost. Therefore, it is a cheap alternative for people or companies that want to create applications without spending large amounts of money on software licenses. Its efficacy makes this language run on low-cost equipment that does not impact the pace of operations in any way.

In general, Fantom is an easy-to-use, cheap, secure, and very readable and writable programming language suitable for many applications. The language supports many programming paradigms and has clear syntax and a rich feature set which makes it an excellent choice for novice and experienced programmers.

Report 2

Variables, Scope, Data Types, Arrays, and Control Statements

Type of Variables

The things that store values in programs are referred to as variables, and they occupy areas of computer memory. They have individual names, each data type and own unique values attached to them. In a computer program, variables hold data that may be retrieved and changed.

Declaration

In other words, when something is assigned as a value within its scope, a variable can later be changed inside that particular scope. Fantom additionally supports type inference for local variables, which eliminates the need for the type signature. If the type signature is absent, then the variable is typed in accordance with the context. The Fantom convention endorses where possible that type infer. Nonetheless, in cases where the type of the right-hand side expression is unclear, it is required to define the type signature of the variable. Initializing a local variable to null frequently results in type inference failing to function. In Fantom, local bindings enable the assignment or creation of a new variable with the same name only in a specified local scope since Fantom also happens to have immutable variables.

Local variables declaration:

```
// syntax
<type> <identifier> := <init>

// example
Str city := “Jeddah”
```

All type names including Void are written in capital letters because Fantom has no primes or primitive keywords. All other methods and variables should begin with small letters. Conventional methods of writing involve placing a camel case on names and not underscores.

Example: aLongVariableName, ALongTypeName

Types may or may not be nullable. It seems that a non-nullable type never stores a null value. A nullable type is indicated by the following "?". This means that non-nullable is the default unless specified otherwise. In Fantom, local variables that are declared without an initial value provided to them follow the same rules as field defaults and unintentionally set to false, 0.0f, null, or 0. This differs slightly from Java or C#, which call for exact assignment.

Data Type

Value types like NET are used by Fantom.
The three special types that are implemented as value types in Java and.NET, respectively, are Float, Int, and Bool. These types all function in the same ways as Boolean, Long, and Double in Java or C#. These kinds, as opposed to Java, logically subclass from Obj to form a class hierarchy. When necessary, the compiler handles boxing and unpacking automatically.

Variable Initialization

The process of initializing a variable—that is, doing so before it is used in any way—is known as the "initialize" technique. Instead of using the more conventional assignment operator (=) to initialize local variables, Fantom employs the (:=) operator. Differentiating between ordinary assignments and local declarations is the primary purpose of the := syntax.

Variables Storage & Lifetime

Static and dynamic storage allocation are both the main types of programming languages.

The compiler allocates a variable's memory at the point of its declaration. This type of memory allocation is also referred to as static memory allocation and compile-time memory allocation. Once an application starts, data variables are assigned to memory. This method is referred to as runtime memory allocation or dynamic memory allocation.

Fantom shares some similarities with Java – both can create dynamic storage allocations. Lifespan of a variable refers to the time for which it stays in memory and gets used when program is run. The lifetime of the variable starts when it is linked to a specific cell and ends when it is no longer linked to the same cell.

Type of Scope

The range to which a name is visible to an object is referred to as its "scope" in computer

languages. It may therefore be brought up in a declaration. Despite the fact that Fantom is statically typed, static scoping can occasionally cause problems for developers. In addition, Fantom guarantees numerous scenarios in which a dynamic system is required. It has some adaptable typing features. The main difference between static and dynamic scoping is that the former depends on the architecture of the program to determine the meaning of a specific variable. Not only that, but a lot of programming languages use this technique. In contrast, dynamic scoping considers the state of the program during runtime. As a result, stack is used to determine the meaning of a given variable.
Here is an illustration of how to use the -> shortcut operator feature of dynamic call:

Java's compile-time verified call is
obj.doStuff()

and runtime reflection is
obj->doStuff()

Note: A method is checked during compilation if it is called with a dot, just like in standard Java. The procedure is called by reflection when the arrow operator is applied. Furthermore, this error can be detected if the method that the reflection is called doesn't exist.

Type of Arrays

As arrays can hold collections of the same type of data, Fantom supports lists and maps. Each item in the list map has its own location. On the other hand, Java Foreign Function Interface, or Java FFI, is a feature of Fantom that automatically uses a list of objects. It attempts to enable programmers to write code in the most efficient manner by using sys::Map syntax for maps and sys::list syntax, which allows us to use lists in fantom. Sadly, functionality for multi-dimensional arrays is still lacking.

Lists

The list will be initialized as follows, showing the objects identified by an Int Declaration in a linear fashion:
list:= ["Abdulrahman," "Ahmed," "Ali"]
Here are some examples of the many slots in Fantom that show you how to manipulate lists:
• Insert

The initial index will be 0 in this example because it demonstrates how to add an element to an empty list:
This function can be used by programmers to add elements to lists using specifications.
list :=[,]

```
list.insert(50 , 8000)
echo(“$list”)
```

• each
individually element in the list receives this function so they can individually carry out a certain task. Here's an example of how to use each element:

```
[“Ali”, “Ahmed”].each | Str s | { echo(s) }
```

• contains
A boolean type is used by the includes method to ascertain if a certain item is included in the list. This is a basic illustration of how it functions:

```
list:=[“Abdulrahman”, “Mohammed”, “Ali”]
echo(list.contains(“ALI”)) // False
```

• isEmpty
To ascertain whether anything is missing from this list. Much like Java's isEmpty function.

• removeAt
After deleting an element from a specified index, the removeAt method returns the removed element. Here's an example:

```
list:=[“Ali”,”Amir”]
echo(list.removeAt(0)) // the list will be [“Amir”]
```

Map

Given a list, map builds another list from it, through application of a particular function across all members of the input list for generating every value in the resulting one.
The list will be declared and initialized as follows:

```
map:= [100:”A+”, 80:”B+”, 70:”C+”]
```

Maps can also be used using specific techniques, like:
• addList
This function adds a certain list to a map to optimize it. In the event that the function is acquired using the keys:

```
map:= [5:”5” , 6:”6”]

m.addList([“7”,”8”])
String type to Int type
m=> [5:”5”, 6:”6” , 7: “7”, 8: “8”]
```

• add
This method adds a value to the current map by providing the key.
map:= [100:"A+", 80:"B+", 70:"C+"]
map.add(60: , "D+") // map:= [100:"A+", 80:"B+", 70:"C+" , 60:"D+"]

• equals
instance:
map_1:= Int:Str[100:"A+", 80:"B+"]
map_2:= Int:Str[100:"A+", 80:"C+"]
map_3:= Int:Str[100:"A+", 80:"B+"]
map_1 == map_2 ===> False
map_1 == map_3 ===> Ture

• vals
Using this technique, a list can be generated from the map. For example:

map:= [5:"5" , 7 :"7"]
map.addList(["5", "9"]) | Str s->Int | { return s.toInt }
echo(map.vals())

Control Statements

Selection statements:

1. If statement:

In Fantom, you can use the 'if' statement to determine whether to execute a specific code block in response to a given condition.

```
if (condition) {
   // rest of code like echo, if, etc.
}
```

Example:

```
Age := 50
if (Age < 80) {
   echo("Age is less than 80")
}
```

2. Switch statement:

You can run different bits of code with the "switch" statement based on the value of a variable. The basic syntax of the "switch" statement:

```
switch (variable) {
   case Case_1: {
      // Your code
   }
   case Case_2: {
      // Your code
   }
case Case_3: {
      // Your code
   }

   default: {
      // Your code if variable doesn't cases, code will be executed
   }
}
```

Example:

```
Age := 50
switch (num) {
   case 60: {
      echo("Age is 60")
   }
   case 55: {
      echo("Age is 55")
   }
   case 50: {
      echo("Age is 50")
   }
   default: {
      echo("Age is not 55, 60, or 50")
   }
}
```

Iterative statements:

1. For loop:

You can iterate through a range of values or a collection of values in Fantom using the 'for' loop. The basic syntax of the 'for' loop is as follows:

```
for (variable in range or collection) {
   // Your code
}
```

Example:

```
for (index in 1..20)
```

```
{
   echo(index)
}
```

2. While loop:

Fantom's "while" loop allows a block of code to be continuously run while a particular condition is true. The basic syntax of the 'while' loop is as follows:

```
while (condition)
{
   // code
}
```

Example:

```
IndexOfNumber := 14
while (IndexOfNumber <= 27) {
   echo(IndexOfNumber)
   IndexOfNumber = IndexOfNumber + 3
}
```

3. Do-while loop:

Fantom’s “do-while” loop permits executing the statement repeatedly when it is still the truth until the statement runs at least once. The fundamental syntax of the 'do-while' loop like this example:

```
do {
   // code
} while (condition)
```

Example:

```
Index := 37
do {
   echo(Index)
   Index = Index + 5
} while (Index <= 105)
```

Code

```
class HelloWorld
{
    static Void main()
    {

    // statment that will display on user screen to enter the number of grades
    Env.cur.out.print ("Enter Number of grades: ").flush
    // decleartion for a var has Int type which will store user input
    Int numGrades
    // storing user input
    try numGrades = Env.cur.in.readLine.toInt
    catch (Err e) // catching error when user enter another type that non-Int
    {
      echo ("You had to enter a number") // statment that will disply on user screen when a TypeError occurred
      return
    }
    //declearing an empty list
    list:=[,]
    // decleartion and initialization for counter
    counter:=1
    // decleartion and initialization for sum var which will add the sum of grades in every iteration the "f" means it holds float numbers
    sum:=0f
   for(i:=0; i<numGrades; ++i){ // for loop will be used numGrades times!
    Env.cur.out.print ("Grade $counter: ").flush // statment that will display on user screen to enter student grade
    Int grade // same comments in line 6-11
     try grade= Env.cur.in.readLine.toInt
        catch (Err e)
    {
      echo ("You had to enter a number")
      return
    }
    list.insert(i,grade) // insert method that take 2 parampters i to specify the index and the next one is the element that will be stored in the list
    sum+=grade // add grade to sum in each iteration
    ++counter // increament the counter
    }
    avg:=sum/numGrades // avg varible that computes the average for the grades
    echo("\nAverage: $avg") // print statment which has new line and to print the average
```

```
Enter Number of grades: 4
Grade 1: 24
Grade 2: 12
Grade 3: 33
Grade 4: 17

Average: 21.5
Maximum Grade: 33
Minimum Grade: 12
```

```
Enter Number of grades: 5
Grade 1: 99
Grade 2: 95
Grade 3: 100
Grade 4: 97
Grade 5: 97

Average: 97.6
Maximum Grade: 100
Minimum Grade: 95
```