14.1

Declaring a constructor private will ensure that no one outside of the class can directly instantiate the class. In this case, the only way to get the instance of the class is by providing a public static method to return an instance of this class. Private constructor is always used in Singleton pattern.

Additionally, as the constructor is private, the class can’t be inherited;

14.2

Yes, the finally block will get executed.

The finally block gets executed no matter the program attempts to exit within try block or catch block. But the finally block will not get executed if the program exits by calling system.exit() or outage.

14.3

Final:

When applied to a class: The class can’t be subclassed.

When applied to a method: The method can’t be overridden in the subclasses.

When applied to a primitive variable: The value of the variable can’t be changed.

When applied to a reference variable: The variable can’t point to other object with the same type.

Finally:

There is an optional finally block after the try block or after the catch block. Statements in the finally block will always be executed (except if JVM exits from the try block). The finally block is used to write the clean code.

Finalize:

Finalize method is called when garbage collector clean up the object.

14.4

1. In C++ templates, the parameters can be any type or integral value, while in java generics, the parameters can only be reference types, not primitive types.

2. In C++ templates, separate copies of the class or function are to be generated for each type parameter when compiled, while in java generics, only one version of the class or function is compiled, works for all type parameters.

3. In C++ templates, objects of a class with different type parameters are different types at run time, while in java generics, type parameters are erased when compiled and objects of a class with different type parameters are the same type at run time.

4. In C++, templates can be specialized -- a separate implementation could be provided for a particular template parameter. While in java, generics cannot be specialized.

5. C++ templates do not support wildcards while java generics support wildcard.

6. C++ templates do not directly support bounding of type parameters, but metaprogramming provides this. While java generics support bounding of type parameters with "extends" and "super" for upper and lower bounds, respectively and allows enforcement of relationships between type parameters.

7. In C++ templates, type parameter of templated class can be used for static methods and variables and static variables are not shared between classes of different type parameters. While in java generics, type parameter of templated class cannot be used for static methods and variables and Static variables are shared between instances of classes of different type parameters.

14.5

Object reflection provides ways to get information about java classes and objects, such us:

1. Getting class type of an object at run time.

2. Creating an instance of a class.

3. Getting information about methods and fields inside a class at run time.

4. Invoking any method of a class at run time.

Object reflection helps in observing or manipulating the runtime behavior of applications and is useful while debugging and testing applications, as it allows direct to access to methods, constructors, fields, etc.

14.6

We can put count variables for put() and get() methods. Whenever they are called, we can increment the count. We can achieve this by extending the existing library map and overriding the put() and get() methods.

If we create multiple instances of map and want to sum up the total count of the times the program calls the put() and get() functions, we can declare the count variables static.