Is Java a Pure Object-Oriented Programming Language?

Luis C. Lopez
The University of Texas at Austin
CS 370 Undergrad Reading and Research

1 Introduction

What is consider a pure Object-Oriented Language? Is Java purely Object-Oriented? There are many papers about the purity of Java as an Object-Oriented Programming language that clarify why in most cases Java is consider "pure" and in others "not pure". In this paper, I will discuss and analyze Java programs to determine how pure Object-Oriented they are.

For a language to be consider Object-Oriented, there are four principals that need to be met: Encapsulation, Data Abstraction, Polymorphism and Inheritance[1]. Certainly, Java is Object-Oriented; it organizes data in a way that lets people do all kinds of things. This is done by the use of objects and classes, which in Java, an object is an instance of a class [2].

2 Object-Oriented Programming

2.1 Encapsulation

Encapsulation in Java is the process of making the fields in a class private and providing access to the fields through public methods [3]. When we create a field to be private, the idea is to hide the implementation details from the users so that it can only be available within the same class. This produces the idea of objects encapsulating data. For the user, the object has a certain behavior and the only way to be able to expose that behavior is by the use of public methods or functions.

2.2 Polymorphism

Polymorphism is one of the most important aspects of Object-Oriented Programming languages. It has the ability for an object to take on many forms, which is the literal definition of polymorphism[2]. Poly means many, and morph means form. The most common use happens when a parent class reference is used to refer to a child class object. For example,

```
public class Animal{}
public class Dog extends Animal{}
…
Dog d = new Dog();
Animal a = d;
Object o = d
```

Figure 2.2: using polymorphism and inheritance

Dog would be consider to be polymorphic due to its multiple inheritance.

2.3 Inheritance

In the previous example about polymorphism, Dog d uses multiple inheritance. Dog, Cat, and Bull, for example, will all share the same characteristics of an Animal. This means that inheritance allows a class to inherit properties of another class. In Figure 2.2, when the Dog class extends to the Animal class, Dog inherits all non-private members including methods and fields. This concept is important in Java and in Object-Oriented Programming languages because the purpose of inheritance is to use polymorphism and to promote code reuse [4].

2.4 Data Abstraction

Programming in Java is based on building data types. When an object is created, that object holds a data type value. In many cases, those data types are abstract; meaning that the user does not need to know how a data type is implemented in order to be able to use it. For example, in a computer we are able to see a monitor, keyboard and mouse but we don't necessarily know anything about the internal wiring. This is abstraction. In programming is the same concept. Data Abstraction allows the user to treat data as it is without understanding how that data is being implemented or calculated [7].

```
Set = int → boolean
        even = func(n) n%2 == 0
        empty = func(n) false
        full = func(n) true
        ...
        …
```

```
Set
        empty: ( ) → Set
        insert: Set int → Set
        member: Set int → boolean
        isEmpty: Set → boolean
        …
```

Figure 2.4: Data abstraction example

In figure 2.4 we can see how on the left hand side we have a set in which the return value for each of the methods is a boolean, but we know that it takes a function func to get a result. On the right hand side, the methods for that Set, either returns a Set or a boolean, the difference this time is that we are unsure about what it is being used to come up with that result. For example, with insert: Set int → Set, it can be any type of container, an array, a list, etc. to come up with the value that is required. The user will only be able to see the result and ignore completely what is happening in reality underneath that function or method.

3 OOPJavaAnalyzer.java

3.1 Java Code Analyzer

Java is a programming language that is considered as pure Object-Oriented language. It incorporates all of the necessary principals to be considered Object-Oriented; it manages encapsulation, polymorphism, inheritance and data abstraction. Also, objects and classes is what makes Java work, without it, any program in Java would not be able to compile. So if objects and classes are necessary and almost everything in the Java programming language is an object, why is it that the question about the purity of the Java language arises? Well, my program OOPJavaAnalyzer.java takes in a Java program as an argument, it checks line by line and it parses the line to check for uses of class as a type like: "instanceof" class, cast to a class, class type passed as a parameter and any other use of class as a type. These uses will inform the user by my Java program analyzer how pure Object-Oriented the Java program is and it will print out all the class types used in the program.

3.2 Uses of class as a Type

A class provides a blueprint for objects; it creates an object by the use of a class [4]. In Java, an object is created by the use of instantiation; it uses the Java operator keyword "new" to create the object.

```
Ex. 1:   Myclass myclass = new Myclass();

Ex. 2:   myclass instanceof Myclass;

Ex. 3:   public void methodExample( Myclass myclass ){}

Ex. 4:   Myclass x = (Myclass) y;

Ex. 5:   Myclass z;
```

Figure 3.2: Examples of uses of class as a Type

In figure 3.2 example 1, "Myclass myclass ..." is the variable declaration that associates a variable name with an object type, with the use of the "new" keyword it creates the object and then it is followed by a call to the constructor, in which it initializes the new object, in this case myclass of type Myclass.

In example 2, instanceof tests to see if myclass is an instance of the class type Myclass; if it is, it will return true (false otherwise). Its most common use is in if-statements, so the way the Java analyzer program counts this type of use in any Java program, is by the use of regular expressions. It will go line by line to check if that string contains the word "instanceof", but it will not count if the string is a line that is equivalent to a System.out.println() statement. Printing out "instanceof" is not the same thing as checking if myclass is an instanceof Myclass.

The last three examples in Figure 3.2 deals with class types passed as a parameter, casting to a class and variable declaration to a class type. These three examples show how an object is being manipulated so that it can be able to communicate within the class. When the the Java program is being analyzed, it will count the number of times each of those uses of class as a type appear in the program. When it finalizes, it will add up the total number of uses of a class as a type and it will be divided by the number of types overall. The result will give the purity of the program.

3.3 Java's OOP purity

I mentioned previously what the main principles are in order for a language to be considered Object-Oriented. We need to have inheritance, polymorphism, encapsulation and data abstraction. Without any of these four principles, if a language is considered to be 100% Object-Oriented, the purity level will decrease and it would not be consider a pure language.

Everything inside a language is an object. Take for example Smalltalk, everything is an object even primitive types [6]. In Java, that is not the case. Many argue the purity of Java as an Object-Oriented Programming language not to be pure. One of the main reasons is due to the fact that primitive data types in Java are not objects. Primitive types like int, bool, float, double, etc. directly contain values. This creates the debate about Java not being pure Object-Oriented. In Java, every variable has a type declared in the code; it can be a primitive data type or a reference data type, in which a reference type is a reference to an object. So then again, if everything in Java has a type declared to a variable, and if primitive data types are not objects but involve more of a data abstraction, then the Java language is Object-Oriented is not 100% pure in some cases.

4 Conclusion

Throughout this paper I talk about Object-Oriented Programming in Java. I explain the four principals that are needed for Java to be Object-Oriented. I also explain my implementation of OOPJavaAnalyzer.java to analyze the purity of the Java program and if is not 100 percent pure, the differences there are between a pure Object-Oriented Programming language to a not so pure Object-Oriented language. Since Java includes primitive data types and uses some sort of data abstraction, can confuse users in thinking that the Java language is a pure Object-Oriented language, but in most cases due to some variables not being declared as objects, then java becomes not pure of an Object-Oriented Programming language. My OOPJavaAnalyzer program can help identify the purity of the program and will show the classes used as types in the entire program.

Acknowledgements

[1] Wallen, Raymond. (July 19, 2005). “4 major principles of Object-Oriented Programming”. Retrieved from http://codebetter.com/raymondlewallen/2005/07/19/4-major-principles-of-object-oriented-programming/

[2] Liberty, J., & Quirk, K. (2004). *Glencoe introduction to Computer Science using Java*. New York, N.Y.: Glencoe/McGraw-Hill.

[3]July 2015. “Java – Encapsulation”. Retrieved from http://www.tutorialspoint.com/java/java_encapsulation.htm

[4] Copyright 1995, 2015. “ What is Inheritance”. Retrieved from https://docs.oracle.com/javase/tutorial/java/concepts/inheritance.html

[5] Sedgewick, R., & Wayne, K. (2011). *Algorithms* (4th ed.). Upper Saddle River, NJ: Addison-Wesley.

[6] William R. Cook: Object-Oriented Programming Versus Abstract Data Types. REX Workshop 1990: 151-178

[7] William R. Cook: On understanding data abstraction, revisited. OOPSLA 2009: 557-572