Java Foundations - Chapter 5 Q&A

Methods & Classes

CSC 110

# Methods

## What are the benefits of writing a method?

Any time you do (or might do) the same thing with different values or variables, that is an indication you may want to write a method using the values or variables as the parameters. Also when you have code that *looks* the same - ask yourself if you could use a method to clear up your code.

## 

## How do I know when to write a method?

Any time you do (or might do) the same thing with different values or variables, that is an indication you may want to write a method using the values or variables as the parameters. Also when you have code that *looks* the same - ask yourself if you could use a method to clear up your code.

# Classes

## What is the purpose of a constructor?

A constructor initializes an object when it is instantiated. Since there may be many ways you want to allow an object to be created, you can overload the constructor to give the client the option. For example, suppose you are writing a voter registration application that keeps track of the voter's name in a Voter class, political party, and the elections they have voted in. You will want to be able to instantiate a Voter object with a name & party like this:

Voter(String name, Party p)

But some voters don’t have a political party preference, so in that case you would want to provide a constructor that only takes a name like this:

Voter(String name)

Then, you can instantiate objects like this:

Voter potus1 = new Voter("George Washington");

Voter potus27 = new Voter("William Howard Taft", REPUBLICAN);

Voter potus29 = new Voter("Thomas Woodrow Wilson", DEMOCRAT);

## 

## What is the purpose of accessor and mutator methods?

Accessor methods (also called getter methods) are used to get information about the state of an object. Suppose you have a Person class that has data members for name, age, and address. The class designer may want to provide accessor methods such as getName() that return the name (as a String), or getAge() that returns the person’s age.

Mutator methods (also called setter methods) are used to *change* the state of an object. So in our Person example, you may want to have a mutator called setAddress(Address new). However, you want to carefully consider what (if any) mutators you provide. Having a set method for each data member is rarely advisable; doing so is essentially the same as making the members public which breaks encapsulation. A better way to allow the state to change is to write methods that encapsulate behavior - so instead of setAge(int age) you could have a method happyBirthday() that increments the age by one (and which checks that it has been a year since the last birthday!).

## What is the purpose of the wrapper classes?

Wrapper classes serve two purposes. First, they allow us to use the primitive types as objects - this is critical for some applications (such as for use with generic classes which we will discuss in a few weeks). Second, they provide useful methods for working with the values of primitive data types, such as converting between types.

## When should we use wrapper classes?

Anytime you need to use one of the primitive types (int, float, char, etc) as an Object, you should use a wrapper class. For example, later this semester we will work with generic collections of Objects. We cannot put an int or float in a collection because only objects are allowed, but we can put an Integer or a Float. The wrapper classes also have helpful static methods, such as methods to convert from one type to another.

## What is the difference between a primitive variable and an object reference variable?

A primitive variable (int, double, etc) holds an actual value while an object reference variable holds a *reference* to a particular value With a primitive variable, the value is stored in the variable’s memory while an object is stored elsewhere and the object reference variable only holds a reference to where that object is stored.

## Why would we use a primitive type over an object?

In Java, objects are more flexible, so in general you should use them. Primitive types tend to be a little more efficient (since they are just a value, not a full object), but you can only add objects to container classes.

## What is this used for?

The "this" reference refers to an object itself. Suppose we have a Thing class. Inside the class definition, we use this to remove ambiguity about what we are talking about when there is a shadowed name. One example is with constructors, such as below:

public ClassExample(String name, int count) {

this.name = name;

this.count = count;

}

Here, the ClassExample has members called name & count, but the constructor also has parameters called name & count. So if we refer to “name” in the body of the constructor method we are referring to the parameter name, not the member name. By using this.name we indicate that we are referring to the member name, not the parameter name.

## Does this refer to an object’s attributes?

Yes. The "this" reference refers to the attributes (data members/properties/etc) of the current object. So while every object has a "this" reference, each "this" reference refers only to the object that owns it. Think of "this" as a first-person pronoun for an object.

## What else is this used for?

The "this" reference can also help remove ambiguity around shadow variables (covered in the next module) and to call a different constructor. For example, we could create constructors as follows:

public ClassExample(String name) {

this(name, 0);

}

public ClassExample(String name, int count) {

this.name = name;

this.count = count;

}

Notice here in the first constructor since no int is passed we call the next constructor with a default value of 0.

## What is the purpose of a toString method?

The toString method is to provide a String representation of an object. The toString method is called implicitly anytime an object is used as a String (such as in a println call). For example, suppose I have a Date class with Month, Day, and Year members. The toString method would be defined so that if I want to print a Date object called today I can do this:

System.out.println("Today is " + today);

You could define such a toString in your Date class as follows:

@Override

public String toString() {

return month + " " + day + ", " + year;

}

When you write a toString method, keep in mind that you do not necessarily need to include everything in the class. Think about what a user would expect the output to be when the toString is called.

## What does it mean that a toString should not modify the state of the object?

Recall that the *state* of an object is the value of its attributes (variables/data members). The purpose of a toString method is to *return a* String *object that represents that state*. If you change the state you are doing more than simply representing that state - and then do you return a String that represents the state before or after your change? As you can see, that complicates things. Moral of the story: don’t change anything about the object in a toString method.

## What does static mean in a method?

The static keyword indicates that a method belongs to the class rather than a specific instance. A static method can be called without instantiating an object. Note that since the method does not belong to a specific instance of the class, it cannot access any non-static class members.

## What is the difference between a static class variable and a non-static class variable?

A static class variable is shared by all objects of that class. So suppose you had a ClubMember class that had a doorCode class variable to store the access code to the clubhouse. If the access code changes, it would need to change for *all* class members so by making it static any changes to the doorCode variable would be propagated to all members. Non-static variables would be unique to each object, but static variables would be shared by all objects of that class.

## What should we create a static class member?

Any member that is applicable to the class *as a whole*, as opposed to objects as individuals is a candidate to be a static member. For example, the Integer class has static members MIN\_VALUE and MAX\_VALUE since those hold for any Integer object and static methods min(a,b) and max(a,b) that calculate the min (or max) of two integers without requiring an Integer object to be instantiated.

## Does Java allow static local variables?

No. Java does not allow static local variables - static variables must be class members. Some languages, such as C++, allow static local variables which limit the scope in which the static variable is visible to just that method, but Java does not.

## What is “abstraction”?

Abstraction is the concept of focusing on the capabilities of an object rather than its specific implementation. For example, if you have a class that is intended to hold a list of names, when using that class you should be focused on things like adding to the list, removing from the list, and printing the list. You should *not* be concerned about whether the list is stored in an array or a giant, comma-separated string, or an ArrayList.

Abstraction is one of the four “Pillars of Object Oriented Programming” (along with encapsulation, inheritance, and polymorphism, all of which we will cover later).

# Advanced Topics

## What is the difference between a public class and a private class?

Public classes are the classes you are already familiar with. Public classes can also contain private classes which are simply classes defined inside the public class which are only available to the class in which they are contained. This is done when the private class is something useful to the public class, but which does not have any use outside that public class. We will see an example of this when we cover Linked Lists - we will create a private node class in the linked list class since the node objects are not useful outside a linked list.

## What other types of programming are there other than Object-Oriented Programming?

There are many other types of programming (called paradigms). The first programming languages that developed were called imperative languages. In those languages, the code would directly manipulate data stored in memory, there was no concept of a “class” - objects were any single entity stored in memory. In recent years, functional programming has become more popular. In functional programming, the function (method) is the primary building block of the languages and the focus is less on processing values than manipulating data (think of mathematical functions). Listing all the other paradigms is outside the scope of this course, but CSC240 goes into the two programming language paradigms above in addition to logic programming with Prolog.

## How can I avoid garbage collection in Java?

Garbage collection is generally considered a Good Thing™ in programming languages as it reduces the need for the programmer to have to constantly do garbage collection themselves. Java is a garbage-collected language, so if you want to avoid garbage collection for performance reasons, you may want to choose a different language such as C++. Anytime you use the new keyword you potentially create garbage that will need to be collected. It is possible to turn off the garbage collector in your Java VM, but that is not recommended in most cases. If you must use Java and you want to avoid garbage collection, you can also write code that produces less garbage by reducing the number of objects you instantiate.

## When would I use a protected member?

Protected members allow child classes to treat members of a parent class as if they were members of the child class. While private members of the parent class are inherited by the child class, the child class cannot access them directly. This makes it easier to write the child class since you don’t have to use accessor/mutator methods to modify parent class members. However, this does break encapsulation, though only a little bit since only child classes can access protected members - protected members are private to everyone else.