Question from class:

Can a method have two return values of different types so that we can use one for errors?

For example:

if (no errors occur)

return correct value;

else

return "An error has occurred";

Answer: No, Java only allows one return type for a method.

Using strings to report errors is a terrible idea as is using special return values (unless the return value can not possibly be mistaken for a real value).

In the above situation, the code that calls the method is expecting a resonable return value and is not expecting a error message. If it receives an error message

it could try to interpret the error as valid data with disasterous results.

Likewise, you should not use print statements to indicate errors unless your routine is also directly reading input typed in the terminal (and is only used with input from the terminal).

Unless the user is actively at the terminal interacting with the program, the user will not see such a message.

There are better ways to handle errors that we will see later in the course.

More on building a class and constructors:

In class we built the Employee class.

Employee's have names, numbers, salaries and positions, and so, following good object-oriented programming style, we create private fields for the data and public getter/setter methods

for the fields.

We did not provide a setter method for number because we felt that an employee should not have its employee number changed. However, setter methods were provided for both name and salary.

Next, we created constructors for the class.

A reminder about constructors:

A constructor is a special method that has no return type and has the same name as the class.

A constructor is called by the new operator after it allocates the space for the instance.

If you do not write a constructor for your class, Java provides a default constructor that takes no input and does nothing(\*).

(\*) actually it does do something, please see below.

Once you write a constructor for your class, you lose the default constructor.

public Constructor(String number) {

this.number = number;

}

Now, we create an instance by providing a single String to new: new Employee("A11-000")

We can create multiple constructors. This is called constructor overloading. Multiple constructors (or methods of the same name) are allowed as long as we change the number and/or types

of the parameters.

A BAD EXAMPLE:

public Constructor(String name) {

this.name = name;

}

This constructor takes a single String just like the one above. Thus we have two constructors that both take a single String, and Java will not allow this.

If we were to call new Employee("A11-000") as above, Java will not know if the input is supposed to be a name or an employee number.

A BETTER EXAMPLE:

public Constructor(String number, String name) {

this.number = number;

this.name = name;

}

Now, the constructor takes two Strings, and we do not already have a constructor that takes two Strings.

public Constructor(String number, String name, int salary) {

this.number = number;

this.name = name;

this.salary = salary;

}

This constructor also as a unique parameter signature, three Strings, and so it is allowed in Java.

Using the constructors:

Given three constructors, we can create an Employee in three different ways:

Employee e1 = new Employee("A11-000");

Employee e2 = new Employee("B13-111", "Harold");

Employee e3 = new Employee("C3P-011", "Malik", 100000);

The constructor that is called in each case is determined by looking at the parameter numbers and types.

Avoiding duplicate code:

Our constructors all have

this.number = number;

Although this code is quite simple, but it will serve as an example for avoiding duplicate code.

Duplicate code creates challenges when we have to improve our programs or fix bugs. We need to remember when we change a line of code to also change everywhere else that code appears.

As a result, we should try to have code only appear in one location though sometimes we can't avoid duplicate code.

In this case, it would be nice to have the constructor that takes three strings call the one that takes two so the constructor that takes two can set the number and name.

Then the constructor that takes three strings only has to set the salary.

Likewise, the constructor that takes two strings could call the one that takes one strings.

Calling another constructor is done through a special method name: this()

this() is just like any other method call except that it is used only to call a constructor in the same class.

Just like with method calls, this() can take input and the specific constructor called is determined by the input to this().

public Constructor(String number) {

this.number = number;

}

public Constructor(String number, String name) {

this(number);

this.name = name;

}

public Constructor(String number, String name, int salary) {

this(number, name);

this.salary = salary;

}

Important Note:

You might be tempted to take this a step further and have the constructors call the getter methods we wrote for Constructor. Doing so would avoid the duplicate assignment statements,

but it has a bad side effect. As we will see in the next class, classes that extend this class can "override" non-private methods thus changing their behavior. If the constructor calls

setSalary() we have no guarantee that the salary field will be set because the setSalary() method may have been overridden.

Thus, if you want a field set in an instance, you must explicitly assign the field in the constructor.

How Constructors Work:

Here are the details of what constructors do:

1) The first line of a constructor must be a call to another constructor. (This is also the only place in the code were we can have a constructor call.)

If you do not explicitly give a constructor call, Java adds a call to the constructor of the parent class that takes no input.

(Note, this is needed because every Employee is an Object, and it needs to be initialized as an Object before we initialize it as an Employee.)

2) The constructors do the following when called:

a) The constructor call that is the first line of the constructor is called.

b) All fields of the instance are initialized.

c) The rest of the constructor body is executed.

Note point (b) above. Java basically takes any assignment statements on your fields and places that code after the constructor call that is the first line of your constructor and

before the rest of the constructor code. This is important to remember for the situations where you care about the order that things are being done in your program.