Review exam 1

Java basics

—primitive types, loops, conditionals
—instance variables, parameters, local variables, static variables
—scope of variables

Instance variable names can clash with parameter names, and local variables. Name clashing is bad style. You can get yourself in a big mess by accidental name clashes. However, in the case that you encounter one, you have to know how to resolve it. The rule is that the variable that is visible at a certain point is the one that’s “closest”. So local variables shadow parameters which shadow instance variables.

—classes and objects; methods, constructors

A class is a pattern. You can think of it as a type.

To create an actual object, it is *not* enough to declare a variable of that class:

```java
SomeClass obj;
```

You have to **create** the object by calling the class constructor (one of them, if there are several).

```java
obj = new SomeClass();
```

Classes usually contain at least one constructor.

You should always call a constructor to create an object, even if your class does not contain any constructor (though it should). You can always call `obj = new SomeClass()` and if the class does not provide one, Java will provide a default one that initializes all instance variables to zero.

You can only call methods on objects. That is, you first have to create an object before you can call methods on it. When you declare an object:

```java
SomeClass obj;
```

this object is **null** (or zero) until you actually create it with the constructor. Calling a method on `obj` will not cause a compile error, but a run time error (null pointer exception, etc).

—this

Use it inside a method. It points to the current object. Usually used when a paramater clashes with an instance variable.

—toString method

Inherited from Object. Classes can override it. Good for debugging. What does the default one print?
Arrays
int[] a = {1, 2, 3, 4};
int[] b;
b = a; // discuss the implications of this

Inheritance
class TwoDPoint{
    int x, y;
    
    public int getX() {return x;}
    public int getY() {return y;}
    ...
}
class ThreeDPoint extends TwoDPoint {
    int z;
    
    public int getZ() {return z;}
    ...
}

ThreeDPoint p;
System.out.println("location is: " + p.getX() + p.getY() + p.getZ());

Type casting
When you call a method, Java checks that the argument you provided matches the
type expected for that method. If the method declared a parameter of type X, Java
accepts an argument of type X, and any sub-class of X, that is, any type that
inherits from X.
class vector {
    ...
    public void add (Object element) {
        ...
    }
    ...
}

Program analysis
We analyze the space (memory) and time of a program. To estimate time we assume
that all instructions/statements are equal, and count the number of instructions
executed by the program on that input.
The time is a function of the size of the input, which we denote by $n$. We are
interested in the worst-case running time of the program on an input of size $n$. We
are interested in the order (rate) of growth of the time, expressed using the Big-Oh notation.

We say \( f(n) \) is \( O(g(n)) \) if there exists a constant \( c > 0 \) such that \( f(n) \leq c \cdot g(n) \forall n > n_0 \).

Examples: \( 3n + 2 \) is \( O(n) \) also \( O(n^2) \), \( O(n \lg n) \). \( 2n^2 + 5n + 7 \lg n + 3 \) is \( O(n^2) \).

We want to express the running time of a program in terms of a common rate of growth: \( O(1), O(\lg n), O(n), O(n \lg n), O(n^2), O(n^3), \ldots \) and so on.

Vectors
Operations. Complexity.

Algorithms
Searching (linear search, binary search).