Deel 1: Totaal Programmeren

Het principe van *totaal programmeren* met je eigen woorden kunnen uitleggen.

In total programming, all possible values will be settled in a 'normal' way. Below you can see an example to understand the concept of total programming more clearly.

Example

In mathematics, a total function is a function which is defined over its entire domain, a partial function is one which has "holes" in its definition. Now, if you have a function which for some input value *v* goes into an infinite recursion or an infinite loop or in general doesn't terminate in some other fashion, then your function isn't defined for v, and thus partial, i.e. not total.

Total programming doesn't allow you to write such a function. All functions always return a result for all possible inputs; and the type checker ensures that this is the case. This simplifies error handling, there aren't simply any.

Instance variables and class variables kunnen uitleggen.

Instance variable in Java is used by Objects to store their states. Variables which are defined without the static keyword and are outside any method declaration are object specific and are known as **instance variables**. They are called so because their values are instance specific and are not shared among instances.

Example

```
class Page {
public String pageName;
// instance variable with public access
private int pageNumber;
// instance variable with private access
```

Class variables are also known as static member variables and there's only one copy of that variable that is shared with all instances of that class. If changes are made to that variable, *all other instances will see the effect of the changes.*

Example

```
public class Product {
    public static int Barcode;
}
```

Value semantics en reference semantics kunnen uitleggen en illustreren met voorbeelden.

Value semantics is a behaviour where variables are copied when assigned to each other or passed as parameters. Primitive types use value semantics.

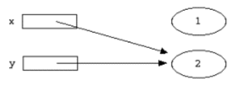
Example

```
int x = 15;
int y = x; // x = 15, y = 15
y = 17; // x = 15, y = 17
x = 18; // x = 18, y = 17
```

Reference semantics is a behaviour where variables refer to a common value when assigned to each other or passed as parameters. Object variables do not store an object; they store the address of an object's location. Reference semantics apply to all class types.

Example 1

```
Point x = new Point(13,12);
Point y = x;
```



Example 2

```
Bankaccount myAccount = null;
final Bankaccount yourAccount = new BankAccount(1323,20);
```

Objects have reference semantics for following reasons:

1. **Efficiency**: Objects can be large and bulky. Having to copy them every time they are passed as parameters would slow down the program.
2. **Sharing**: Since objects hold important state, it is often more desirable for them to be shared by parts of the program when they're passed as parameters. Often we want the changes to occur to the same object.

De basisprincipes voor de verificatie van classes kunnen uitleggen en toepassen.

Zie p.109-120