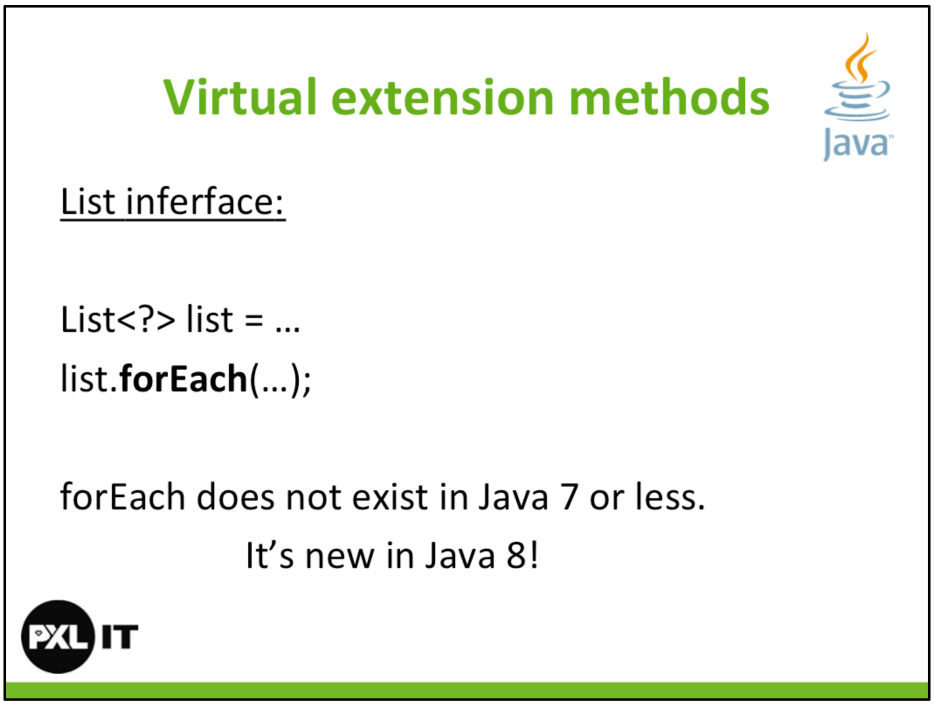
W2: Default methods (Java)

.NET / Java

DE HOGESCHOOL
MET HET NETWERK

Hogeschool PXL – Dep. PXL-IT – Elfde-Liniestraat 26 – B-3500 Hasselt
www.pxl.be - www.pxl.be/facebook



Virtual extension methods

List inferface:

```
List<?> list = …
list.forEach(…);
```

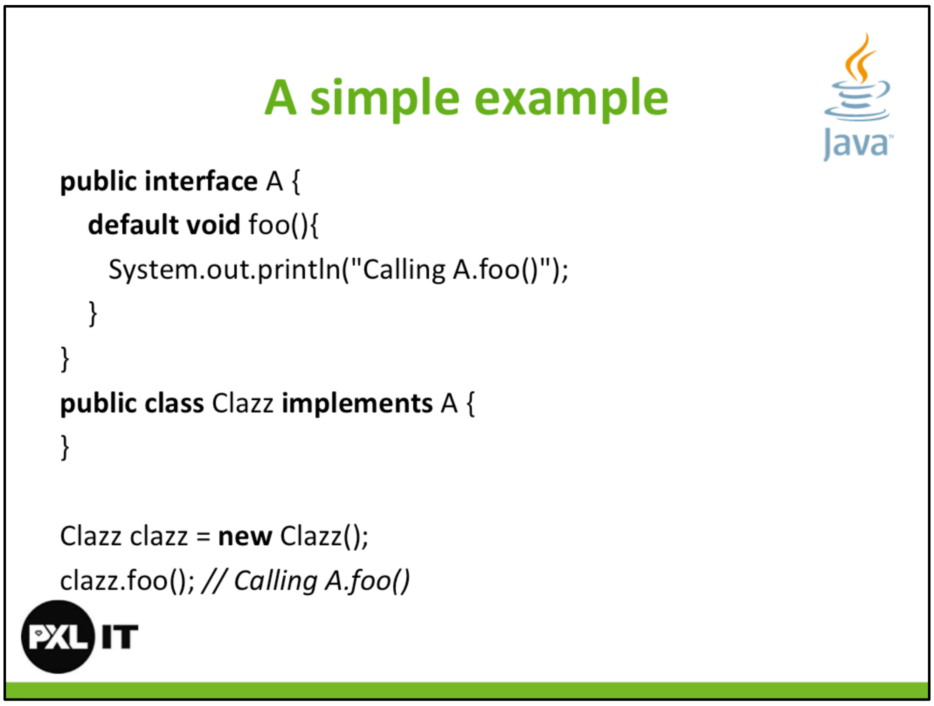
forEach does not exist in Java 7 or less.

It's new in Java 8!

The forEach isn't declared by java.util.List nor the java.util.Collection interface yet. One obvious solution would be to just add the new method to the existing interface and provide the implementation where required in the JDK. However, once published, it is impossible to add methods to an interface without breaking the existing implementation.

Due to the problem described above a new concept was introduced. Virtual extension methods, or, as they are often called, defender methods, can now be added to interfaces providing a default implementation of the declared behavior.

Simply speaking, interfaces in Java can now implement methods. The benefit that default methods bring is that now it's possible to add a new default method to the interface and it doesn't break the implementations.

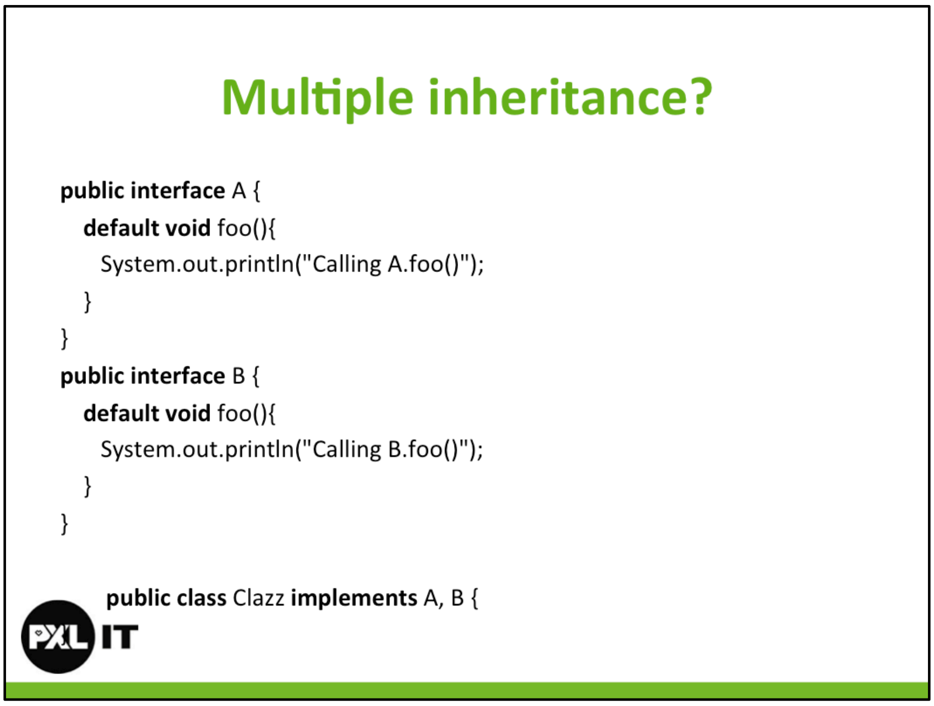


A simple example

```
public interface A {
   default void foo(){
      System.out.println("Calling A.foo()");
   }
}
public class Clazz implements A {
}

Clazz clazz = new Clazz();
clazz.foo(); // Calling A.foo()
```

The code compiles even though Clazz does not implement method foo(). Method foo() default implementation is now provided by interface A.

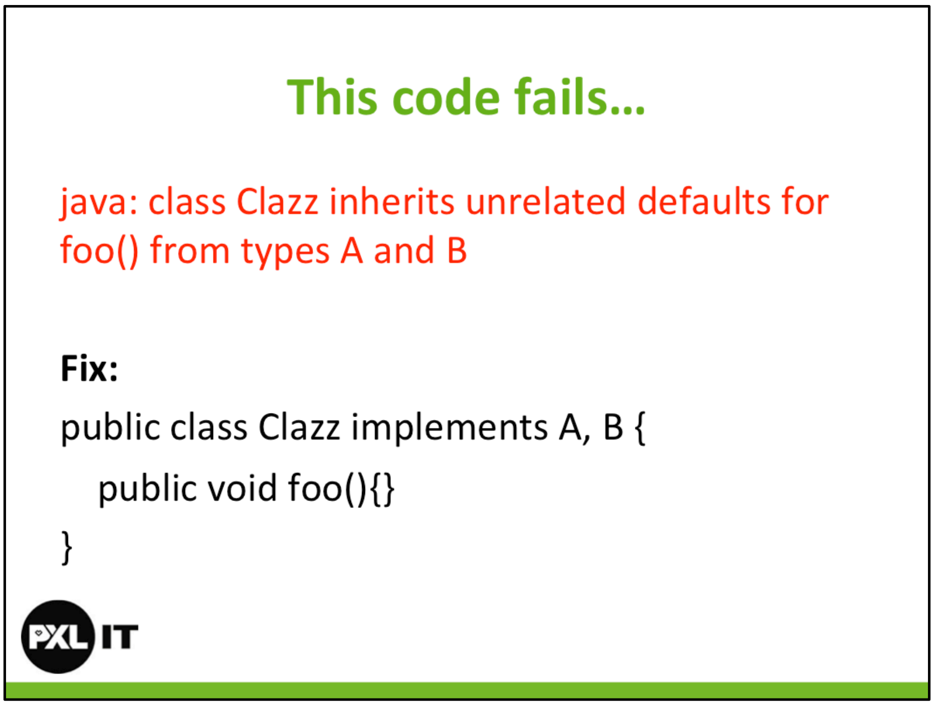


Multiple inheritance?

```
public interface A {
   default void foo(){
     System.out.println("Calling A.foo()");
   }
}
public interface B {
   default void foo(){
     System.out.println("Calling B.foo()");
   }
}

public class Clazz implements A, B {
```

There is one common question that people ask about default methods when they hear about the new feature for the first time: *"What if the class implements two interfaces and both those interfaces define a default method with the same signature?"*.



This code fails…

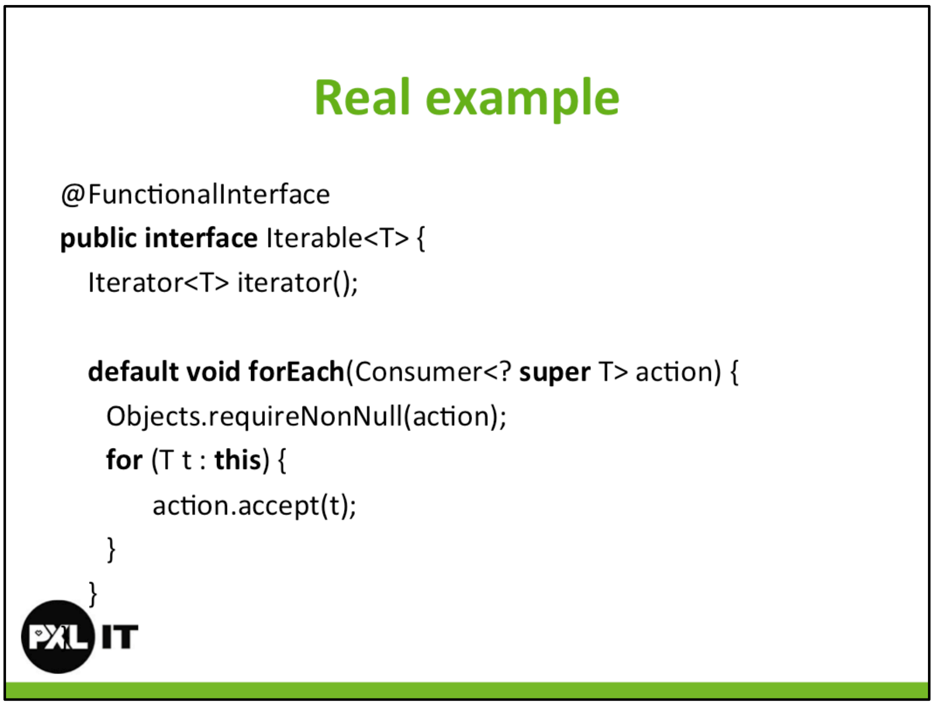
java: class Clazz inherits unrelated defaults for foo() from types A and B

Fix:

```
public class Clazz implements A, B {
   public void foo(){}
}
```

But what if we would like to call the default implementation of method foo() from interface A instead of implementing our own. It is possible to refer to refer to A#foo() as follows:

```
public class Clazz implements A, B {
   public void foo(){
     A.super.foo();
   }
}
```

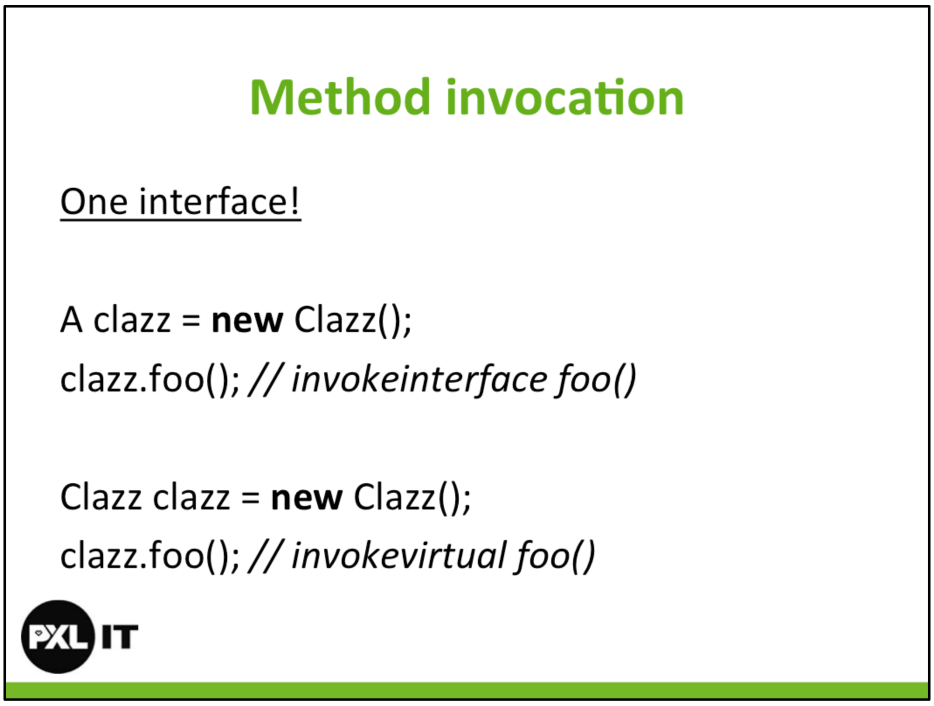


Real example

```
@FunctionalInterface
public interface Iterable<T> {
  Iterator<T> iterator();

  default void forEach(Consumer<? super T> action) {
    Objects.requireNonNull(action);
    for (T t : this) {
        action.accept(t);
    }
  }
```

http://stackoverflow.com/questions/4343202/difference-between-super-t-and-extends-t-in-java



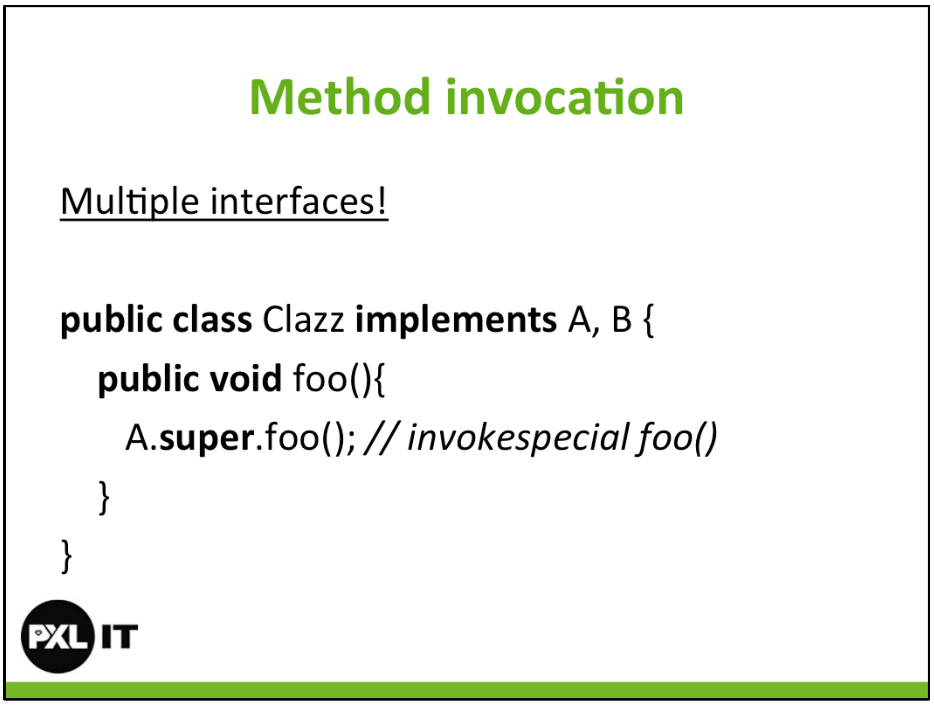
Method invocation

One interface!

```
A clazz = new Clazz();
clazz.foo(); // invokeinterface foo()

Clazz clazz = new Clazz();
clazz.foo(); // invokevirtual foo()
```

From the client code perspective, default methods are just ordinary virtual methods. Hence the name – virtual extension methods. So in case of the simple example with one class that implements an interface with a default method, the client code that invokes the default method will generate invokeinterface at the call site.

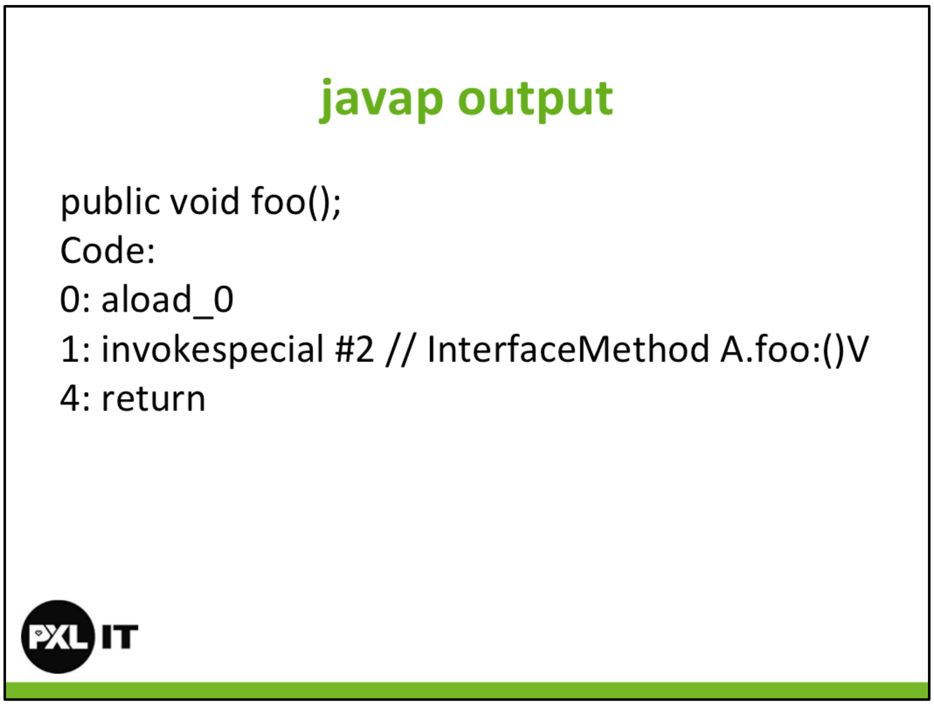


Method invocation

Multiple interfaces!

```
public class Clazz implements A, B {
   public void foo(){
     A.super.foo(); // invokespecial foo()
   }
}
```

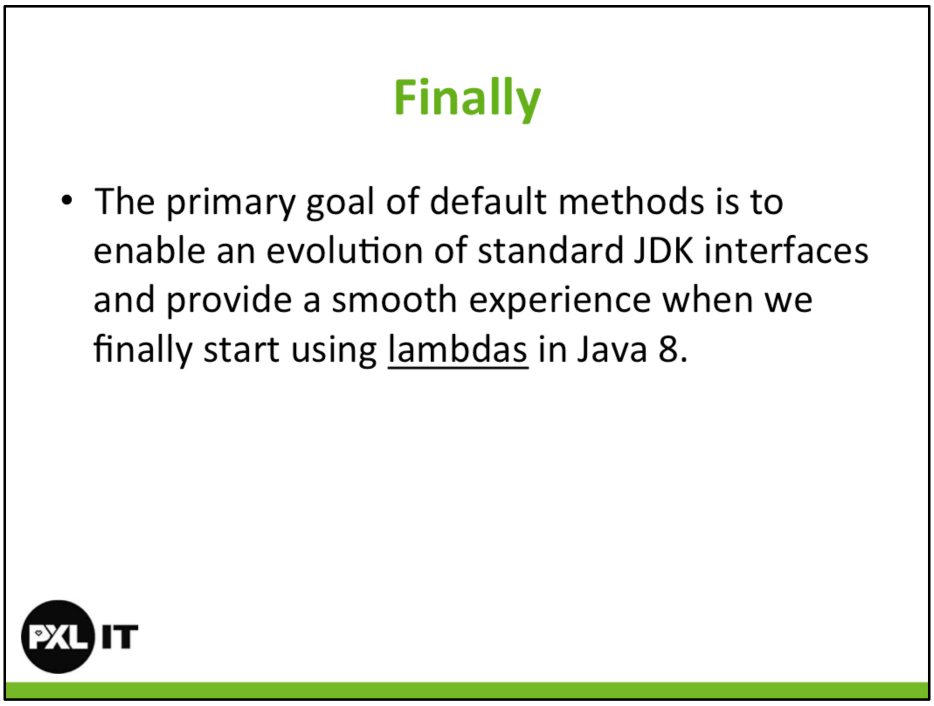
In case of the default methods conflict resolution, when we override the default method and would like to delegate the invocation to one of the interfaces the invokespecial is inferred as we would call the implementation specifically:



javap output

```
public void foo();
Code:
0: aload_0
1: invokespecial #2 // InterfaceMethod A.foo:()V
4: return
```

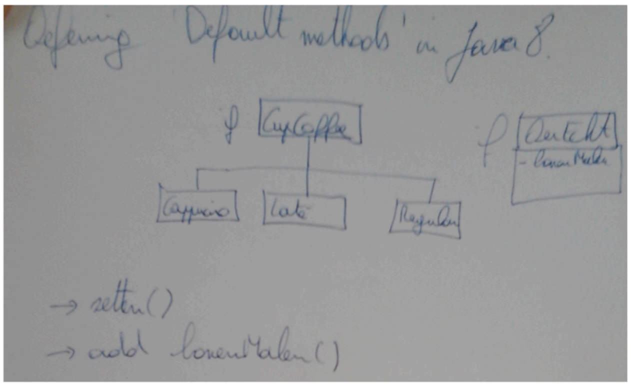
As you can see, invokespecial instruction is used to invoke the interface method foo(). This is also something new from the bytecode point of view as previously you would only invoke methods via super that points to a class (parent class), and not to an interface.



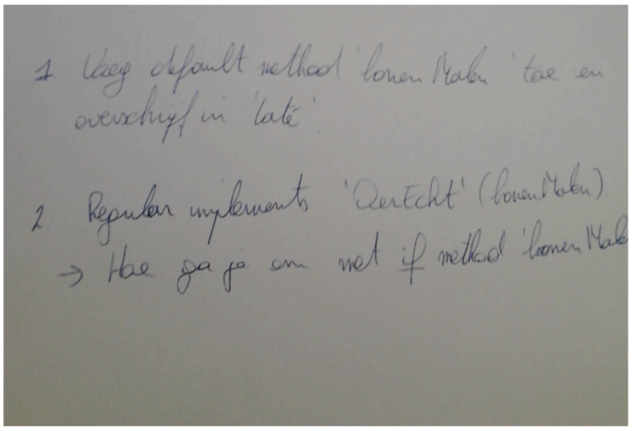
Finally

- The primary goal of default methods is to enable an evolution of standard JDK interfaces and provide a smooth experience when we finally start using lambdas in Java 8.

Excercise


Defining 'Default methods' in Java 8.
I CupCoffee
Cappucino
Latte
Regular
- LemonMaker
→ setter()
→ add LemonMaker()

Excercise



1. Voeg default method 'lonenMaken' toe en overschrijf in 'Late'.

2. Regular implements 'QerEcht' (lonenMaken)
→ Hoe ga je om met if method 'loonenMak