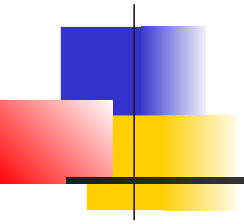


Interface "New" Features

Many new capabilities were added to interfaces in Java 8.

Methods with Code

- Default methods
  - instance methods can have method code!
  - class "inherits" the default implementation

- Static methods with code

  Interfaces can have static methods, which must have an implementation.

Default Method in Interfaces

Add a getCurrency method to Valuable with default implementation that returns "Baht".

```
public interface Valuable {
    /** abstract method - the typical use */
    public double getValue();

    /** default method for getCurrency() */
    default public String getCurrency() {
        return "Baht";
    }

    // ILLEGAL: Cannot override an existing method
//  default String toString() { return "error!": }
}
```

Static Method in Interface

A static **`create()`** method - create money.

Implementation must be included in the interface.

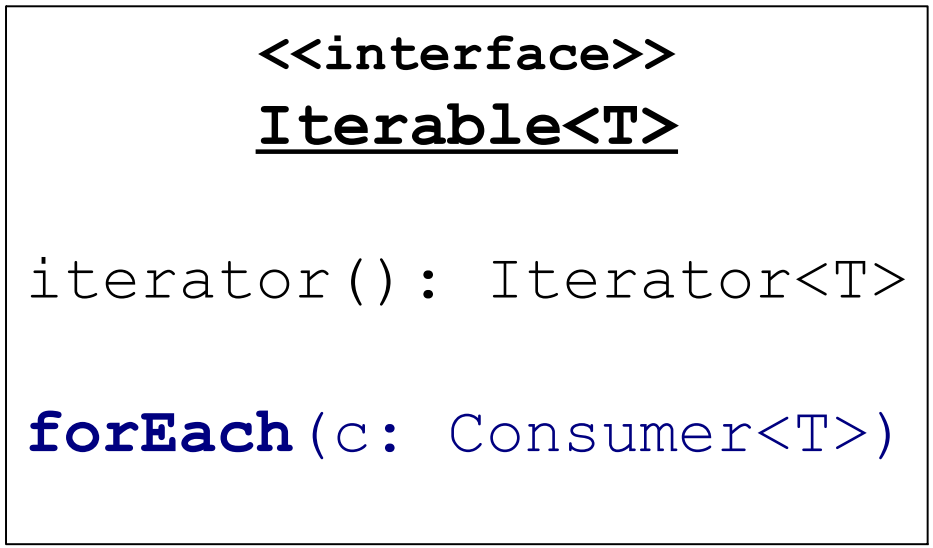
```
public interface Valuable {
    /** value of this item. This is abstract. */
    public double getValue();

    /** Create money (static). public by default */
    static Valuable create(double value) {
        return MoneyFactory.getInstance()
                           .createMoney(value);
    }
}
```

Why add default methods?

Java added new methods to existing interfaces.

How to do that without breaking existing applications?

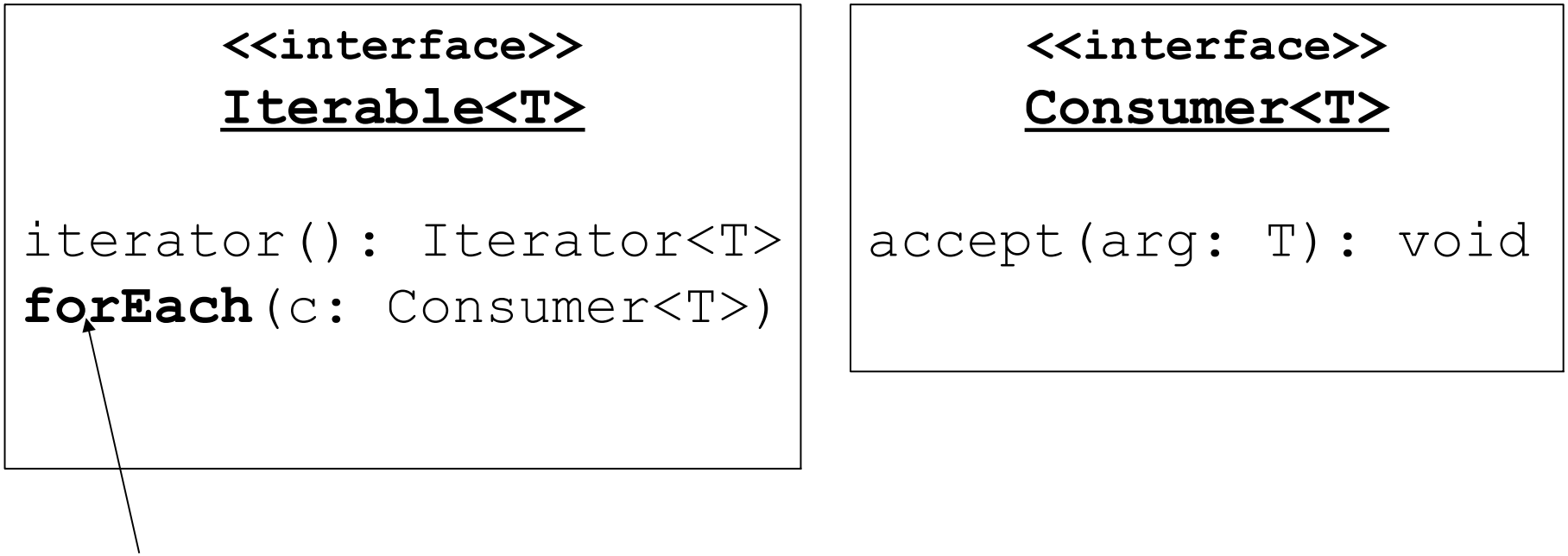
Example: `Iterable` has a new "`forEach`" method:



← New method

"default" method preserves old code

The Iterable interface has a default forEach() method that invokes a Consumer object with each element of the Iterable.



Example: Print each Student in a List

Instead of a loop: use `forEach` and a `Consumer`

```
List<Student> classlist =
                Registrar.getStudents("01219116");
// Anonymous Class to print a student
Consumer<Student> printer = new Consumer<Student>()
{
     public void accept(Student student) {
          System.out.println(studnet);
     }
};
// Use Consumer to print each student in List
classlist.forEach( printer );
```

Let's make the code shorter using a Lambda expression...

Using forEach with Lambda

A Lambda (anonymous function) can implement an interface that has only one method:

```
List<Student> classlist =
                Registrar.getStudents("01219116");
// Consumer as Lambda Expression
Consumer<Student> printer =
                    (s) -> System.out.println(s);
// print each student
classlist.forEach( printer );
```

Let's make this code even shorter using a Method Reference.

forEach with Method Reference

The Lambda just calls System.out.println(s) with the same parameter (s), so we can refer to "println" directly.

"ClassName::methodName" is a *method reference*.

```
List<Student> classlist =
                Registrar.getStudents("01219116");
// Consumer as Method Reference
Consumer<Student> printer = System.out::println;

// print each student
classlist.forEach( printer );
```

Method reference is so clear and concise we can use it inline:

```
classlist.forEach( System.out::println );
```

References

The Java Tutorial:
https://docs.oracle.com/javase/tutorial

Java 8 Features with Examples

http://www.journaldev.com/2389/java-8-features-with-examples

Has short examples of several features