Von der Schleife zum Iterator

Sven Eric Panitz

Hochschule Rhein-Main
Standort Wiesbaden

```java
int i = 1;

while (i<=10) {
   System.out.println(i*i);
   i = i + 1;
}
```

```
int i = 1;

while (i<=10) {
   System.out.println(i*i);
   i = i + 1;
}
```

```java
int i = 1;

while (i<=10) {
   System.out.println(i*i);
   i = i + 1;
}
```

```java
int i = 1;

while (i<=10) {
   System.out.println(i*i);
   i = i + 1;
}
```

```java
int i = 1;

while (i<=10) {
   System.out.println(i*i);
   i = i + 1;
}
```

```java
int i = 1;

while (i<=10) {
   System.out.println(i*i);
   i = i + 1;
}
```

```java
int i = 1;
while (i<=10) {
   System.out.println(i*i);
   i = i + 1;
}
```

```
int i = 1;
while (i<=10) {
   System.out.println(i*i);
   i = i + 1;
}
```

➔ Initialisiere

```
int i = 1;
while (i<=10) {
   System.out.println(i*i);
   i = i + 1;
}
```

- Initialisiere
- Teste auf weiteren Durchlauf

```
int i = 1;
while (i<=10) {
   System.out.println(i*i);
   i = i + 1;
}
```

- Initialisiere
- Teste auf weiteren Durchlauf
- Schalte weiter

```
int i = 1;
while (i<=10) {
   System.out.println(i*i);
   i = i + 1;
}
```

- Initialisiere
- Teste auf weiteren Durchlauf
- Schalte weiter
- Schleifenrumpf

```java
int i = 1;
while (i<=10) {
   System.out.println(i*i);
   i = i + 1;
}
```

- Initialisiere
- Teste auf weiteren Durchlauf
- Schalte weiter
- Schleifenrumpf

```java
int i = 1;
while (i<=10) {
   System.out.println(i*i);
   i = i + 1;
}
for(                                    ){


}
```

```
int i = 1;
while (i<=10) {
   System.out.println(i*i);
   i = i +1;
}
for(int i = 1;                    ){

}
```

```java
int i = 1;
while (i<=10) {
   System.out.println(i*i);
   i = i +1;
}
for(int i = 1;i<=10;                    ){

}
```

```java
int i = 1;
while (i<=10) {
   System.out.println(i*i);
   i = i +1;
}
for(int i = 1;i<=10;i = i + 1){

}
```

```java
int i = 1;
while (i<=10) {
   System.out.println(i*i);
   i = i +1;
}
for(int i = 1;i<=10;i = i + 1){
   System.out.println(i*i);
}
```

```
for(int i = 1;i<=10;i = i + 1){
   System.out.println(i*i);
}
```

```
for(int i = 1;i<=10;i = i + 1){
   System.out.println(i*i);
}

public interface Loop {



}
```

```
for(int i = 1;i<=10;i = i + 1){
   System.out.println(i*i);
}

public interface Loop {
   boolean test();


}
```

```
for(int i = 1;i<=10;i = i + 1){
   System.out.println(i*i);
}


public interface Loop {
   boolean test();
   void step();

}
```

```
for(int i = 1;i<=10;i = i + 1){
   System.out.println(i*i);
}


public interface Loop {
   boolean test();
   void step();
   void body();
}
```

```
public interface Loop {
   boolean test();
   void step();
   void body();
}
```

```
public interface Loop {
  boolean test();
  void step();
  void body();
}
```

```
public interface Loop {
  boolean test();
  void step();
  void body();
}
```

```
public interface Loop {
   boolean test();
   void step();
   void body();
}

abstract class AbstractLoop<E> implements Loop{




}
```

```
public interface Loop {
  boolean test();
  void step();
  void body();
}

abstract class AbstractLoop<E> implements Loop{
  public E current;



}
```

```
public interface Loop {
  boolean test();
  void step();
  void body();
}

abstract class AbstractLoop<E> implements Loop{
  public E current;
  public AbstractLoop(E current) {
    this.current = current;
  }
}
```

```
abstract class AbstractLoop<E> implements Loop {
  E current;
  public AbstractLoop(E current) {
    this.current = current;
  }
}
```

```
public class Counter
        extends AbstractLoop<Integer>{
```

```
public class Counter
        extends AbstractLoop<Integer>{

  public Counter(Integer c) {
    super(c);
  }
```

```
public class Counter
        extends AbstractLoop<Integer>{
  public Counter(Integer c) {
    super(c);
  }
  @Override
  public boolean test() {
      return current <= 10;
  }
```

```
public class Counter
        extends AbstractLoop<Integer>{
  public Counter(Integer c) {
    super(c);
  }
  @Override public boolean test() {
    return current <= 10;
  }
  @Override public void step() {
     current = current + 1;
  }
```

```
public class Counter
        extends AbstractLoop<Integer>{
  public Counter(Integer c) {
    super(c);
  }
  @Override public boolean test() {
    return current <= 10;
  }
  @Override public void step() {
    current = current + 1;}
  @Override public void body() {
     System.out.println(
       current * current);
   }
```

```
public class Counter
         extends AbstractLoop<Integer>{
  public Counter(Integer c) {
    super(c);
  }
  @Override public boolean test() {
    return getCurrent() <= 10;
  }
  @Override public void step() {
    current = current + 1;
  }
  @Override public void body() {
    System.out.println(current * current);
  }
}
```

```
for ( Loop l = new Counter(1)
      ; l.test()
      ; l.step()) {
   l.body();
}
```

```
Loop loop = new Counter(1);

for ( ;loop.test(); loop.step()) {
  loop.body();
}
default void runLoop() {
   for (; test(); step()) {
      body();
   }
}
```

```
Loop loop  = new Counter(1);

loop.runLoop();
```

java.util

Interface Iterator<E>

Method Summary

All Methods | Instance Methods | Abstract Methods | Default Methods

Modifier and Type	Method and Description
boolean	**hasNext**() Returns true if the iteration has more elements.
E	**next**() Returns the next element in the iteration.

```java
import java.util.Iterator;
public class Bis10 implements Iterator<Integer>{
  int i;
  public Bis10(int i) {
    this.i = i;
  }
  @Override public boolean hasNext() {
    return i<=10;
  }
  @Override public Integer next() {
    return i++;
  }
}
```

```
public static void main(String[] args) {

  for ( Iterator<Integer> l  = new Bis10(1)
      ; l.hasNext()
      ;                    ){
       int i = l.next();
       System.out.println(i*i);
  }
}
```

All Methods | Instance Methods | Abstract Methods | Default Methods

Modifier and Type	Method and Description
default void	forEachRemaining(Consumer<? super E> action) Performs the given action for each remaining element until all elements have been processed or the action throws an exception.

```
public static void main(String[] args) {
  Iterator<Integer> loop = new Bis10(1);

  loop.forEachRemaining
       (i -> System.out.println(i));
}
```

`public interface Iterable<T>`

Implementing this interface allows an object to be the target of the "for-each loop" statement. See **For-each Loop**

Method Summary

All Methods	Instance Methods	Abstract Methods	Default Methods

Modifier and Type	Method and Description
`Iterator<T>`	`iterator()` Returns an iterator over elements of type T.

```
Iterable<Integer> iter = () -> new Bis10(3);

for (Integer i :  iter){
  System.out.println(i*i);
}
```