Controlling Program Flow

‘for’ statements

Loops: for

- The *for* loop has two variants: the old-style “basic *for* loop” and the new style “enhanced *for* loop” (also known as the *for-in* or *for-each* loop).

- The basic *for* loop is more flexible than the enhanced *for* loop, but the enhanced *for* loop was designed to make it easier to iterate through arrays and collections.

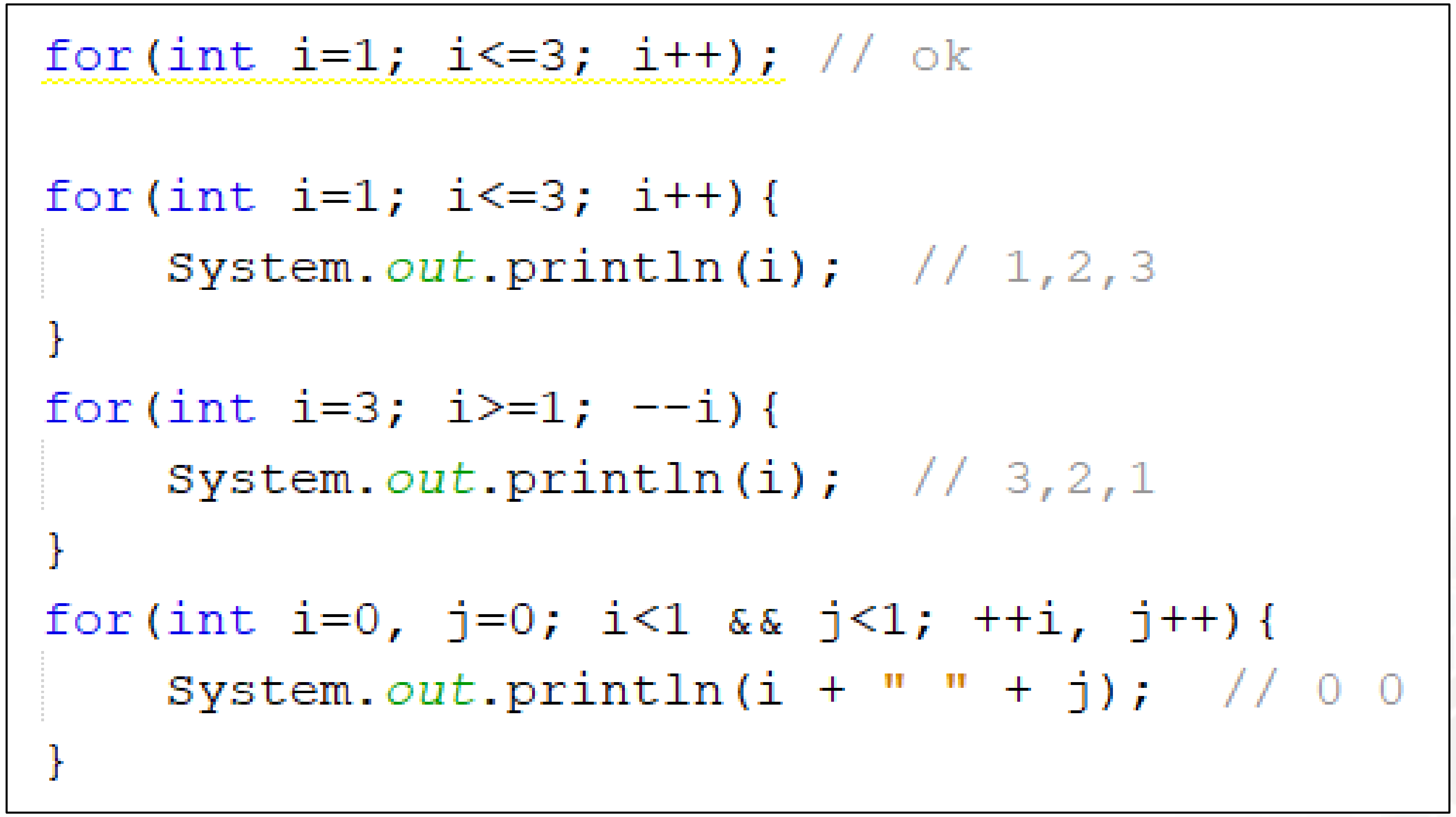
Loops: basic ‘for’

- The general syntax is:

```
for(initialisation; booleanExpr; incr/decr/update){
   // do something
}
```

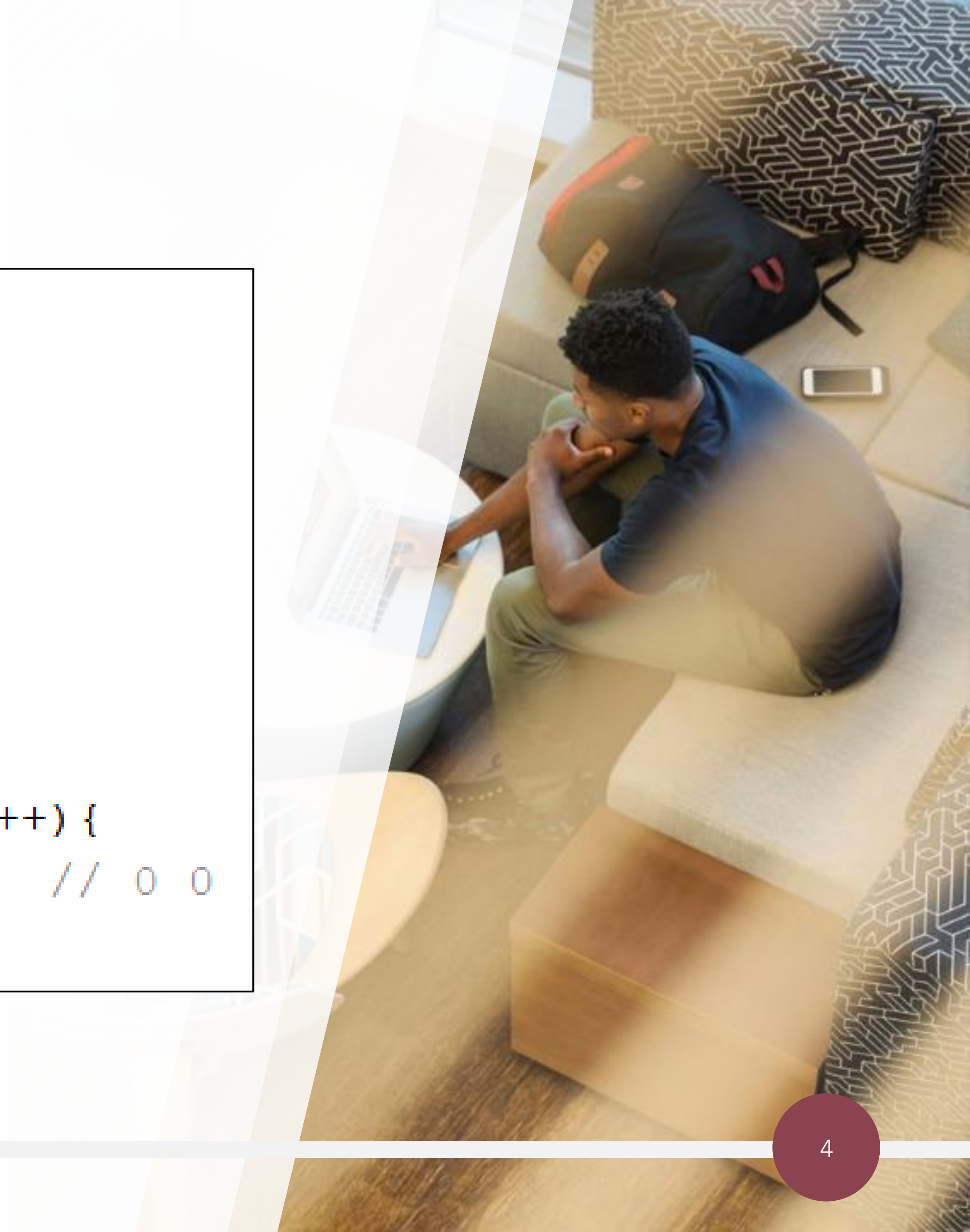
- {} required if the loop controls more than 1 statement.
- Do not re-declare a variable in the initialisation section.
- The variables in the initialisation section must be all of the same type.
- Watch out for infinite loops and scope!

Loops: basic ‘for’

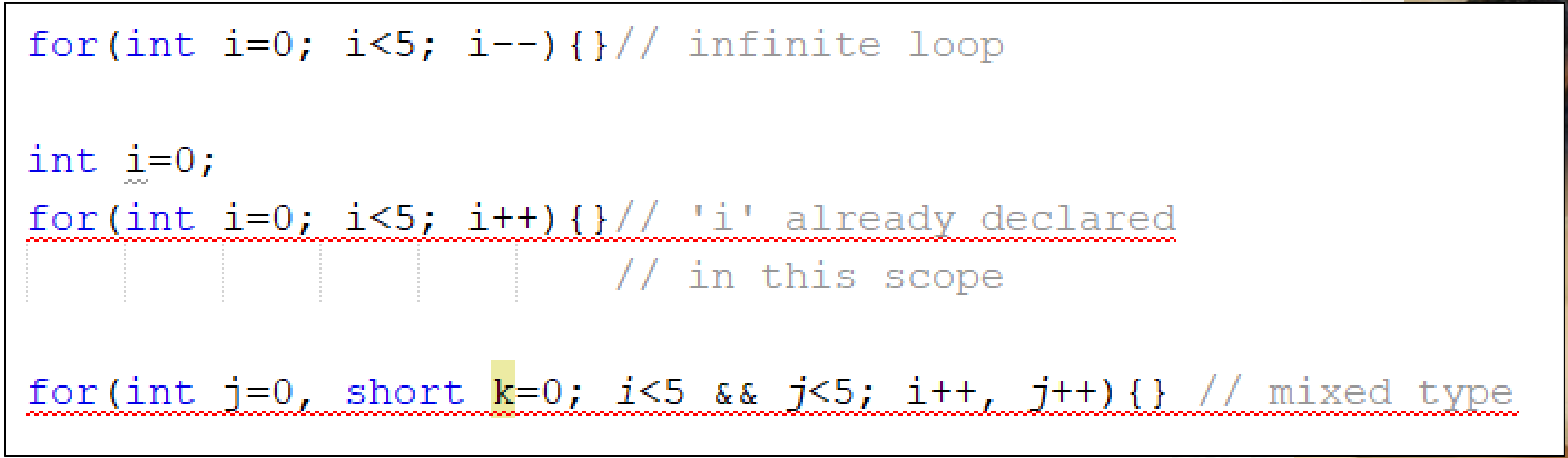


```
for(int i=1; i<=3; i++); // ok

for(int i=1; i<=3; i++){
    System.out.println(i);  // 1,2,3
}
for(int i=3; i>=1; --i){
    System.out.println(i);  // 3,2,1
}
for(int i=0, j=0; i<1 && j<1; ++i, j++){
    System.out.println(i + " " + j);  // 0 0
}
```



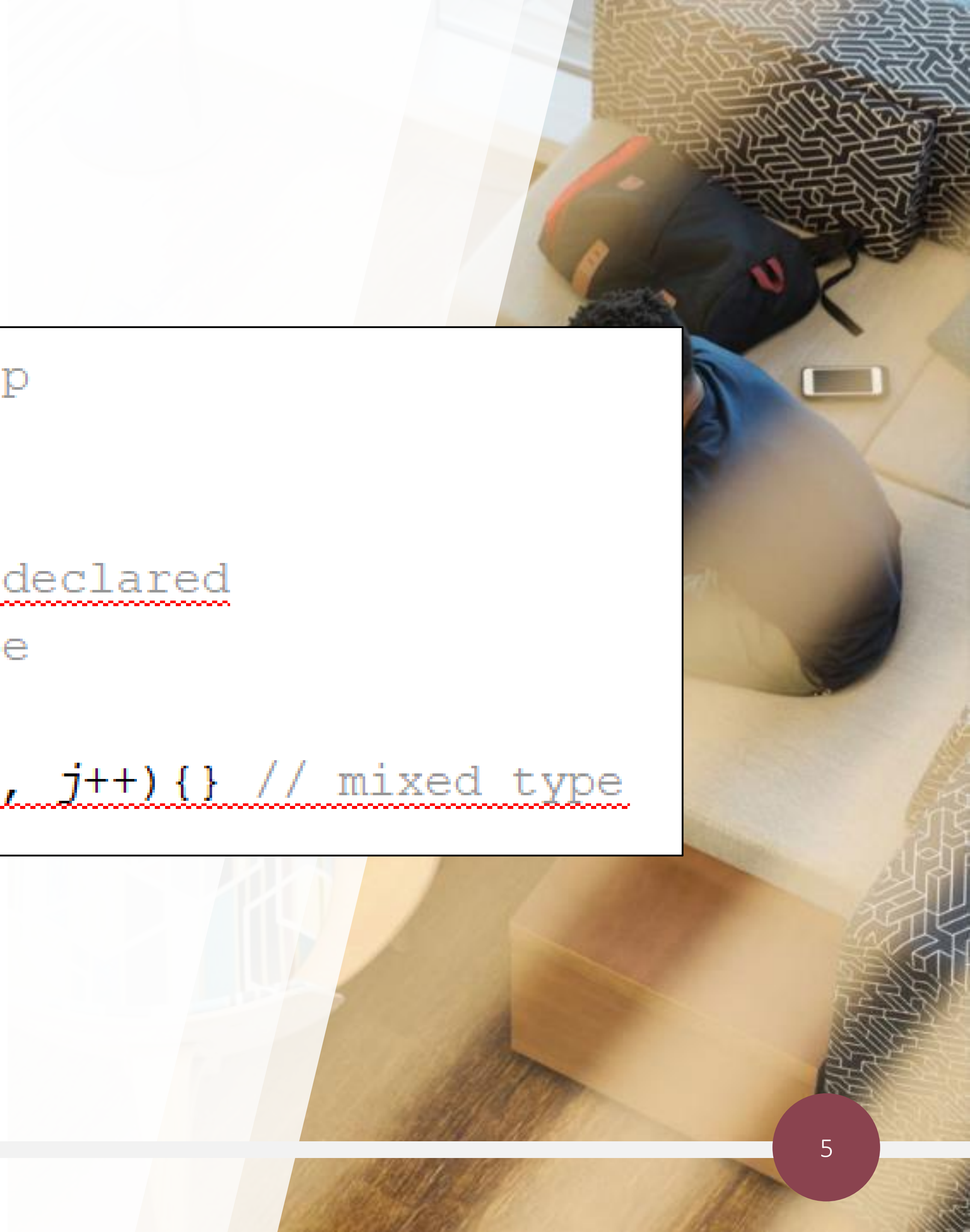
Loops: basic ‘for’



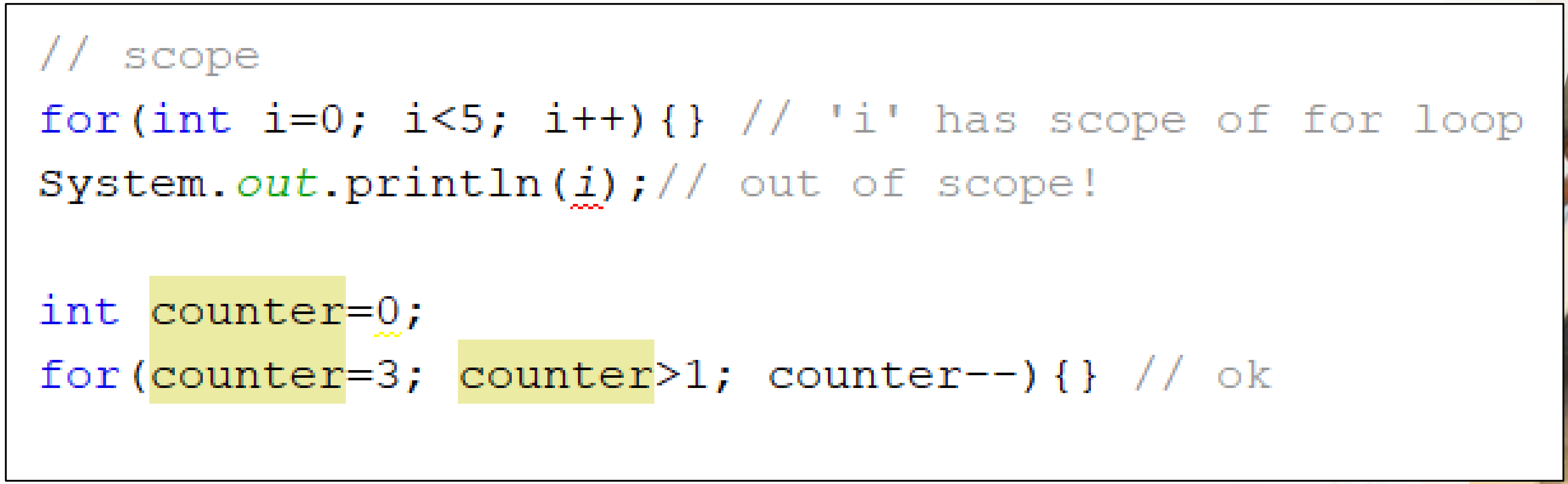
```
for(int i=0; i<5; i--){}// infinite loop

int i=0;
for(int i=0; i<5; i++){}// 'i' already declared
                        // in this scope

for(int j=0, short k=0; i<5 && j<5; i++, j++){} // mixed type
```

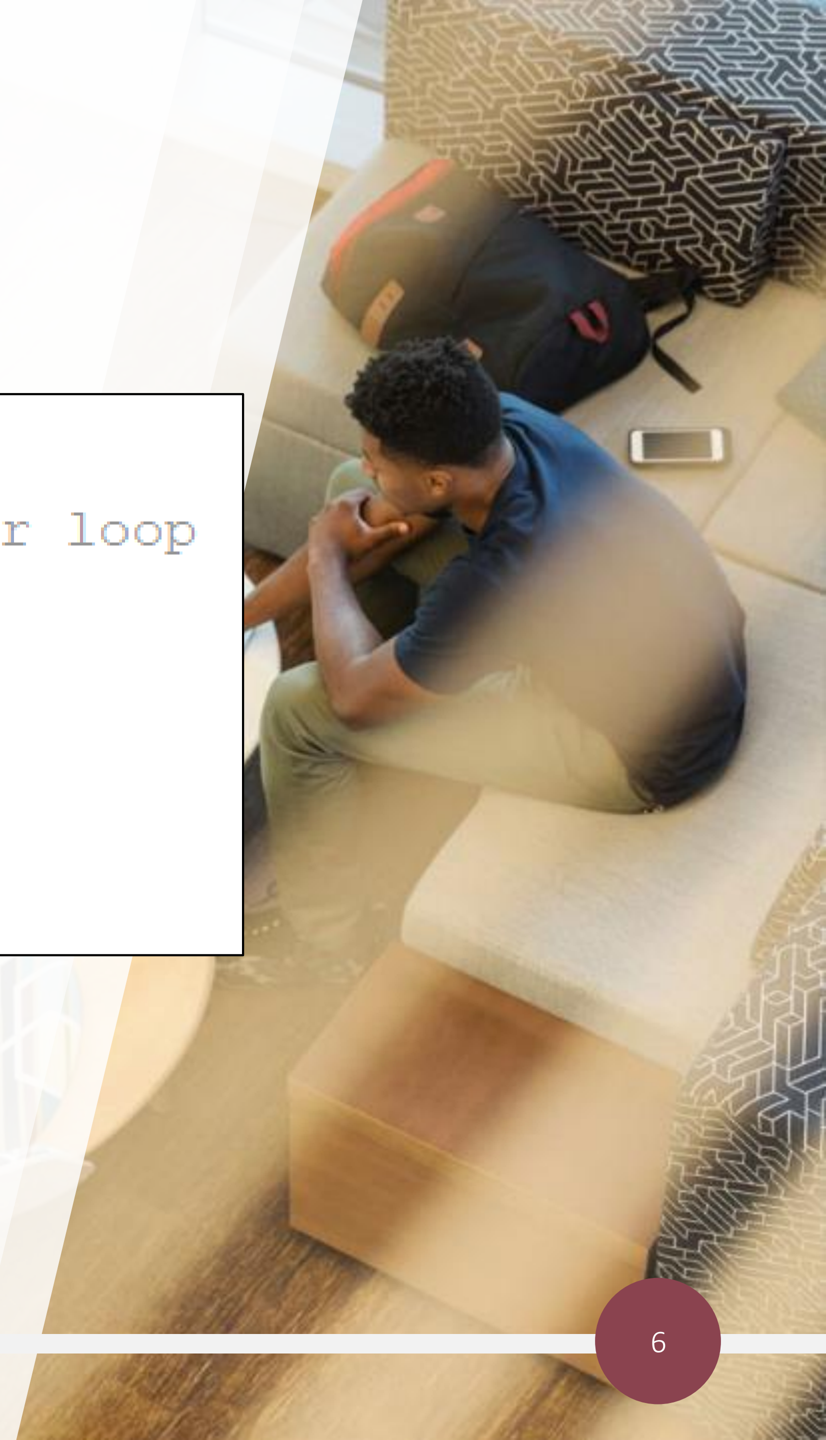


Loops: basic ‘for’



```
// scope
for(int i=0; i<5; i++){} // 'i' has scope of for loop
System.out.println(i);// out of scope!

int counter=0;
for(counter=3; counter>1; counter--){} // ok
```



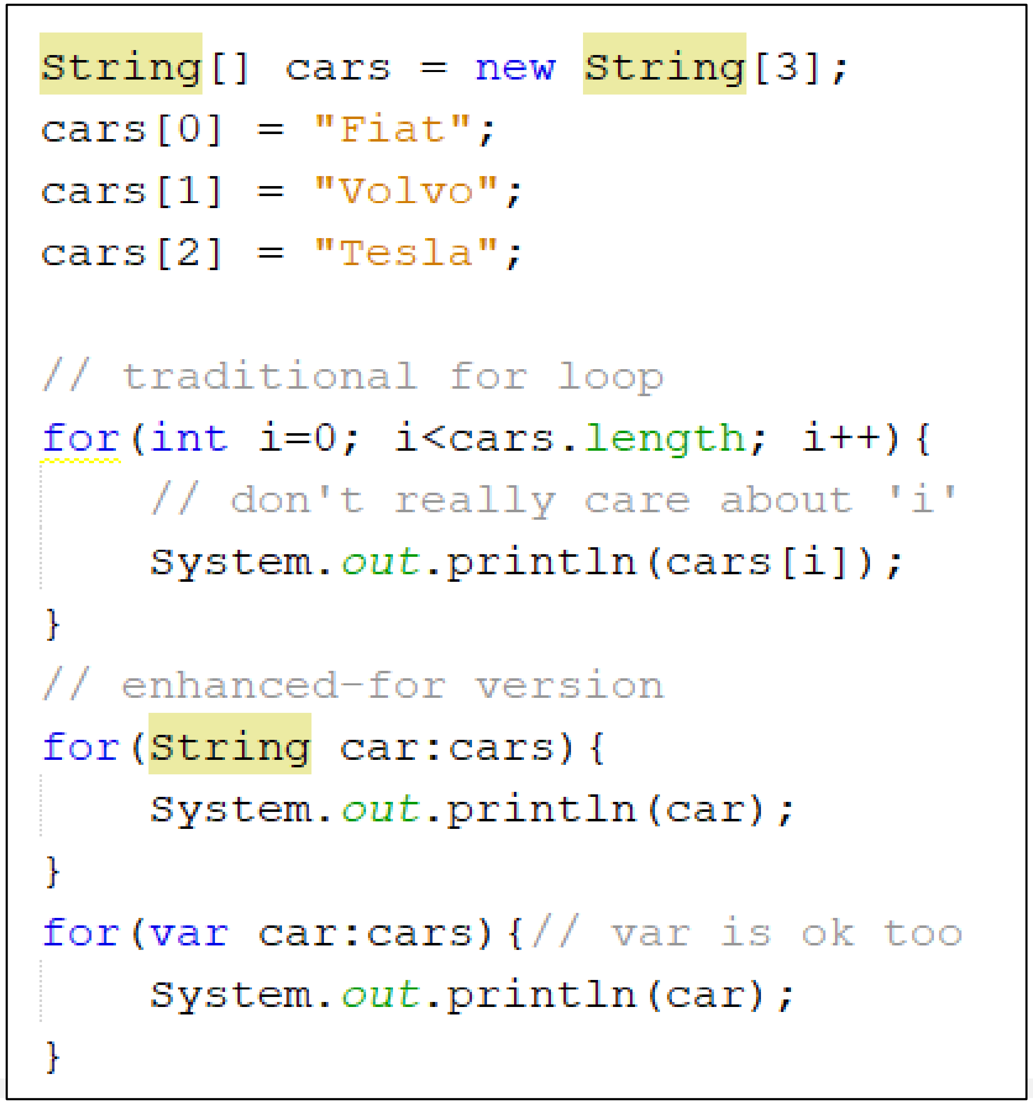
Loops: ‘enhanced for’

- Enhanced *for* loop (used in iterating through arrays/collections)

```
    for(datatype variableName : array/collection){
        // code
    }
```

- variable declaration – the *newly declared* block variable has the scope of the loop; its type will be compatible with the elements in the array you are accessing; its value is the current array element (this will change obviously)
- array/collection – the array can be any type e.g. primitives or objects; the collection (i.e. *Iterable*), for example, a *List* or *Set*.

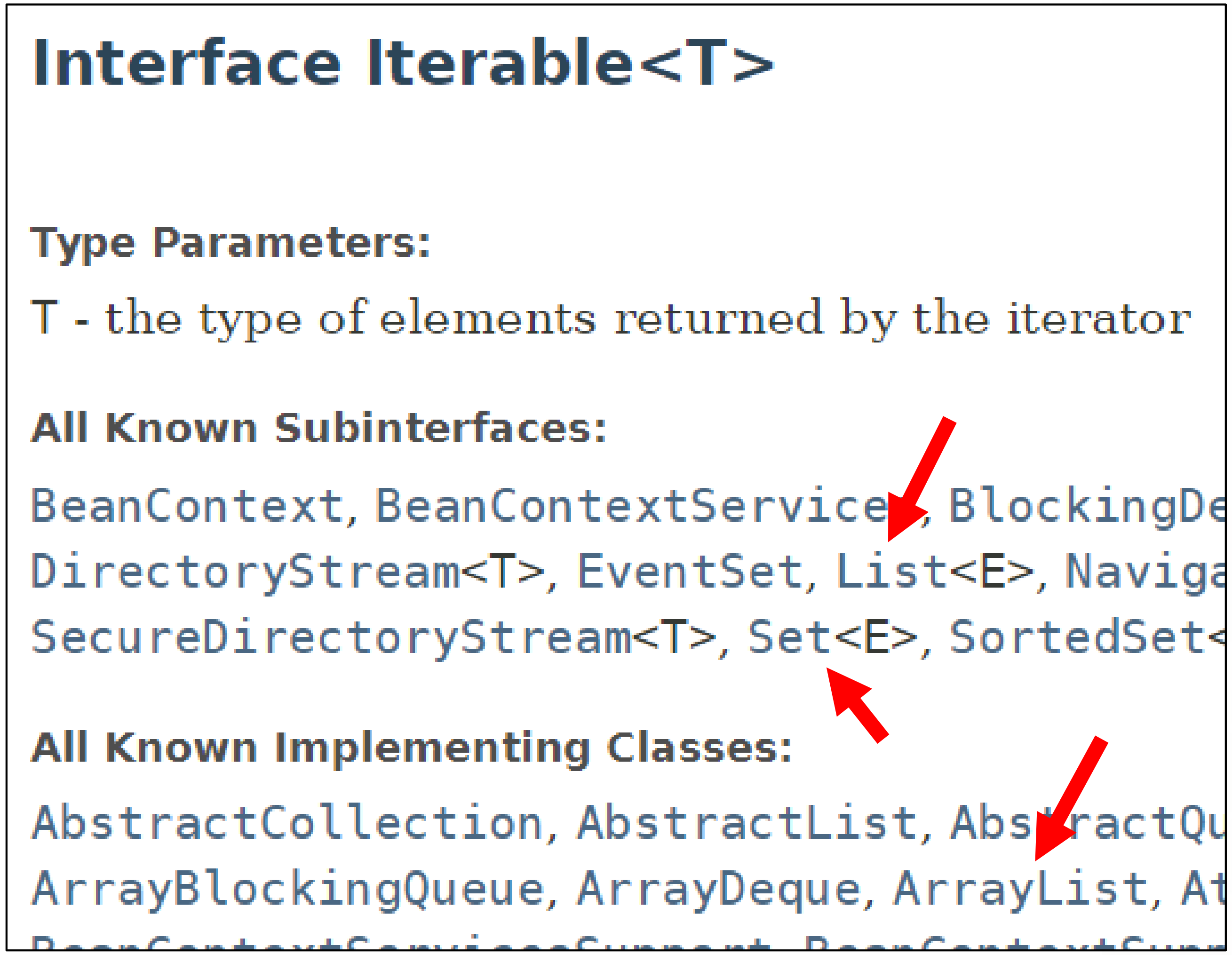
Loops: ‘enhanced for’



```
String[] cars = new String[3];
cars[0] = "Fiat";
cars[1] = "Volvo";
cars[2] = "Tesla";

// traditional for loop
for(int i=0; i<cars.length; i++){
    // don't really care about 'i'
    System.out.println(cars[i]);
}
// enhanced-for version
for(String car:cars){
    System.out.println(car);
}
for(var car:cars){// var is ok too
    System.out.println(car);
}
```

Loops: 'enhanced for'



Interface Iterable<T>

Type Parameters:

T - the type of elements returned by the iterator

All Known Subinterfaces:

BeanContext, BeanContextServices, BlockingDe
DirectoryStream<T>, EventSet, List<E>, Naviga
SecureDirectoryStream<T>, Set<E>, SortedSet<

All Known Implementing Classes:

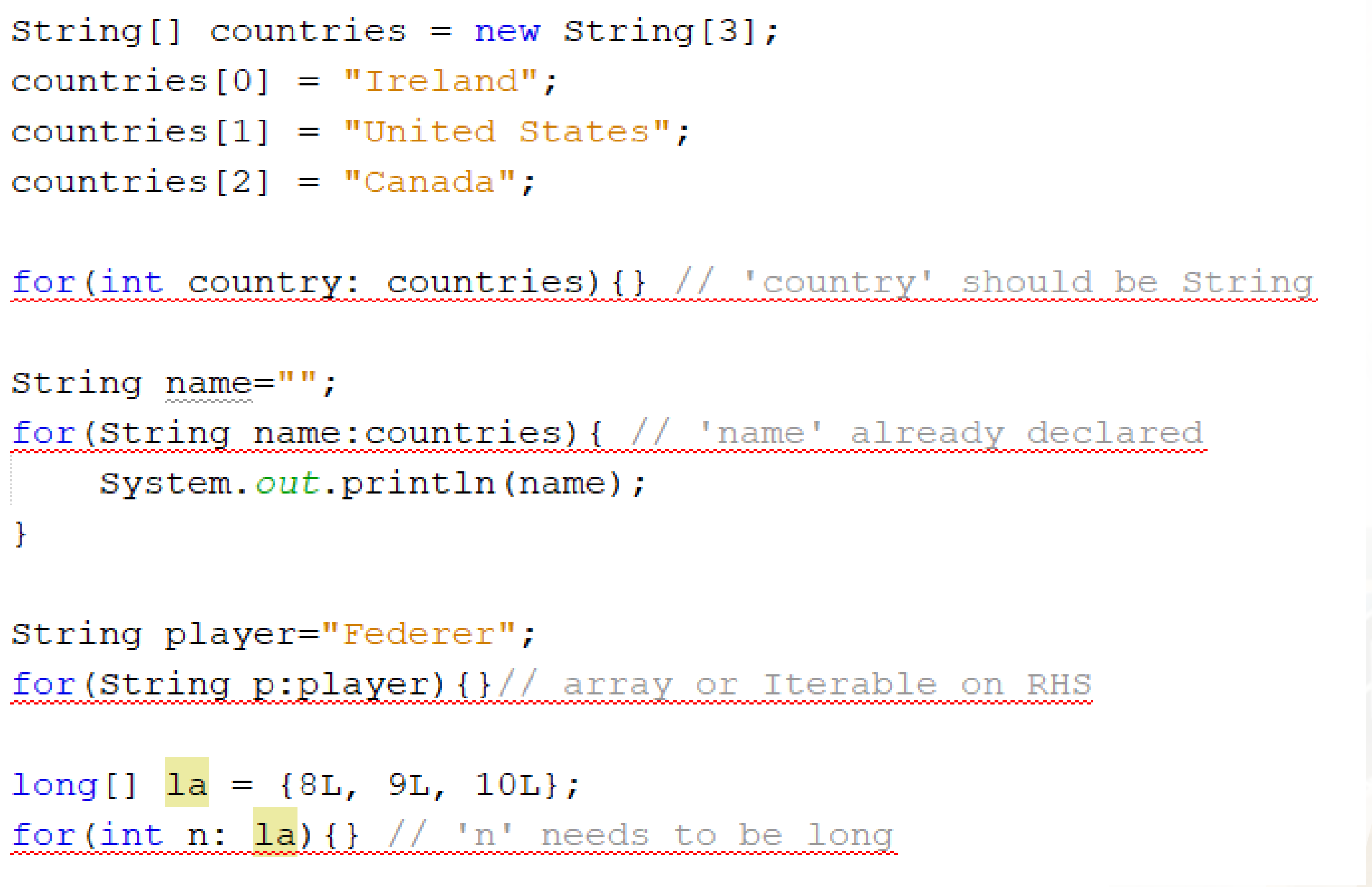
AbstractCollection, AbstractList, AbstractQu
ArrayBlockingQueue, ArrayDeque, ArrayList, At

Loops: ‘enhanced for’

```
List<String> cars = new ArrayList<>();
cars.add("Fiat");
cars.add("Volvo");
cars.add("Tesla");

// enhanced-for version - using an Iterable
for(String car:cars){
    System.out.println(car);
}
```

Loops: ‘enhanced for’



```
String[] countries = new String[3];
countries[0] = "Ireland";
countries[1] = "United States";
countries[2] = "Canada";

for(int country: countries){} // 'country' should be String

String name="";
for(String name:countries){ // 'name' already declared
    System.out.println(name);
}

String player="Federer";
for(String p:player){}// array or Iterable on RHS

long[] la = {8L, 9L, 10L};
for(int n: la){} // 'n' needs to be long
```