

# Lists revisited

# ArrayLists

An ArrayList is a collection of objects of the same type. It is:

- ordered
- indexed
- variable length

# Type parameters

When we create a variable that holds an ArrayList we also need to provide the **type of objects** it contains:

```
ArrayList<String> listOfStrings;
```

```
ArrayList<Integer> listOfInts;
```

```
ArrayList<Book> listOfBooks;
```

These are called the **type parameters**.

# Type parameters

When you create a variable that holds an  
Array, you also need to provide the **type of  
objects** it contains:

collection  
class



```
ArrayList<String> listOfStrings;
```

```
ArrayList<Integer> listOfInts;
```

```
ArrayList<Book> listOfBooks;
```

These are called the **type parameters**.

# Type parameters

When we create a **collection class** that holds an array of **item class** objects, we also provide the **type of objects** it contains:

```
ArrayList<String> listOfStrings;
```

```
ArrayList<Integer> listOfInts;
```

```
ArrayList<Book> listOfBooks;
```

These are called the **type parameters**.

# Type parameters

When we create a **collection class** that holds an **item class**, we also provide the **type of objects** it contains:

```
ArrayList<String> listOfStrings;  
ArrayList<Integer> listOfInts;  
ArrayList<Book> listOfBooks;
```

These are called the **type parameters**.

angle  
brackets

# Lists of ints

Note that we can't have an ArrayList containing a **primitive type** (e.g. ints, booleans).

Instead we use **object-wrappers** (Integer, Boolean).

Java lets us freely **convert** between the two.

```
Integer objWrap = 77;
```

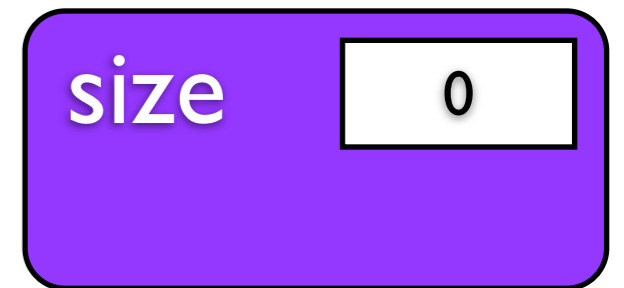
```
int i = objWrap;
```

# Construction

To construct an ArrayList we will usually use the simple no-parameter constructor:

```
// create an empty list
```

```
ArrayList<String> classRoll =  
    new ArrayList<String>();
```



There are two other constructors but we won't be using them much.

# Adding items

A newly constructed list is **empty**.

We can add items using the **add** method.

```
classRoll.add("Malcolm");  
classRoll.add("Claude");
```

size	2
0	"Malcolm"
1	"Claude"

The type of objects we add must **match** the type parameter.

```
classRoll.add(47); // ERROR
```

# Adding duplicates

We can add something to a list more than once.

```
classRoll.add("Malcolm");  
classRoll.add("Malcolm");
```

size	4
0	"Malcolm"
1	"Claude"
2	"Malcolm"
3	"Malcolm"

# Size

We can use the `size` method to access the number of objects on the list:

```
int nStudents = classRoll.size();  
// nStudents == 2  
classRoll.add("Sim");  
nStudents = classRoll.size();  
// nStudents == 3
```

size	2
0	"Malcolm"
1	"Claude"

# Size

We can use the `size` method to access the number of objects on the list:

```
int nStudents = classRoll.size();  
// nStudents == 2  
classRoll.add("Sim");  
nStudents = classRoll.size();  
// nStudents == 3
```

size	3
0	"Malcolm"
1	"Claude"
2	"Sim"

# Get

We can access elements on the list by their index using the `get` method.

```
String me = classRoll.get(0);
```

```
// me == "Malcolm"
```

```
String him = classRoll.get(1);
```

```
// him == "Claude"
```

# Indexing

Note that the indices always **start from 0** and end at size-1.

```
classRoll.size();  
// returns 3  
classRoll.get(0);  
// returns "Malcolm"  
classRoll.get(2);  
// returns "Sim"  
classRoll.get(3); // ERROR
```

size	3
0	"Malcolm"
1	"Claude"
2	"Sim"

# Remove

We can use the `remove` method to remove the element at a given index.

The other elements get **renumbered**.

```
classRoll.remove(1);
```

```
classRoll.size();
```

```
// returns 2
```

```
classRoll.get(1); // "Sim"
```

size	3
0	"Malcolm"
1	"Claude"
2	"Sim"

# Remove

We can use the `remove` method to remove the element at a given index.

The other elements get **renumbered**.

```
classRoll.remove(1);
```

```
classRoll.size();
```

```
// returns 2
```

```
classRoll.get(1); // "Sim"
```

size	2
0	"Malcolm"
1	"Sim"

# Other methods

There are other methods you can read about in the [API documentation](#):

- **clear** - remove all elements
- **set** - set value of a given element
- **insert** - insert an element at an index
- **contains** - test if an element is on the list

# Loops

Often in our code we want to perform a sequences of statement **multiple times**.

```
System.out.println(classRoll.get(0));  
System.out.println(classRoll.get(1));  
System.out.println(classRoll.get(2));  
System.out.println(classRoll.get(3));
```

Loop statements allow us to **abbreviate** this to a single statement.

# Loops

There are several different kinds of loops:

- **for-each loops** - do an operation on each item in a collection
- **while loops** - do an operation until a condition changes
- **for-loops** - do an operation a specified number of times

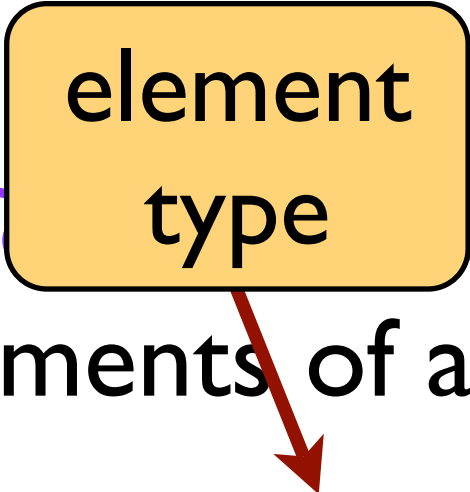
# For-each loop

A **for-each loop** is used to iterate over all the elements of a collection:

```
for (String who : classRoll) {  
    // action performed on  
    // each element  
  
    System.out.println(who) ;  
}
```

# For-each loop

element  
type



A **for** loop is used to iterate over all the elements of a collection:

```
for (String who : classRoll) {  
    // action performed on  
    // each element  
    System.out.println(who);  
}
```

# For-each loop

element  
type

element  
variable

A **for** loop is used to iterate over all the elements of a collection:

```
for (String who : classRoll) {  
  
    // action performed on  
    // each element  
  
    System.out.println(who) ;  
  
}
```

# For-each loop

element  
type

element  
variable

list  
variable

A **for** loop is used to iterate over all the elements of a collection:

```
for (String who : classRoll) {  
  
    // action performed on  
    // each element  
  
    System.out.println(who) ;  
  
}
```

# For-each loop

If the list is **empty**, the for-each loop will do nothing.

```
classRoll.clear()  
  
for (String who : classRoll) {  
    // this never runs  
    System.out.println(who);  
}
```

# While loop

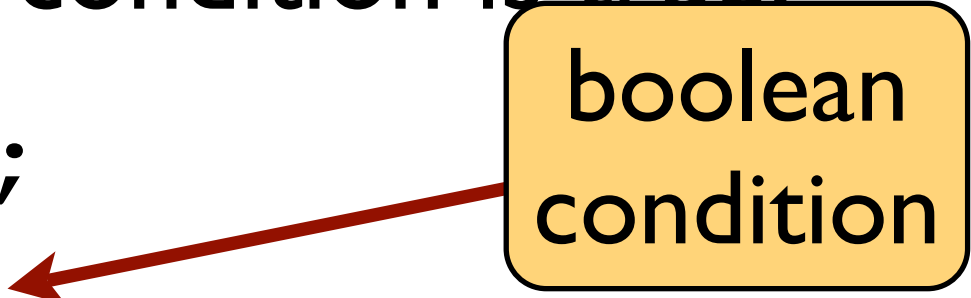
A **while loop** is used keep performing some action as long as some condition is true.

```
int angle = 735;
while (angle >= 360) {
    angle -= 360;
}
// angle == 735-360-360 == 15
```

# While loop

A **while loop** is used keep performing some action as long as some condition is true.

```
int angle = 735;  
while (angle >= 360) {  
    angle -= 360;  
}  
// angle == 735-360-360 == 15
```



boolean  
condition

# While loop

If the condition is already **false**, the while loop will do nothing.

```
int angle = 90;
while (angle >= 360) {
    angle -= 360;
}
// angle == 90
```

# For loop

A **for loop** is used to do something a given number of times.

```
for (int i = 0;  
    i < 5;  
    i = i + 1) {  
    // done 5 times  
    System.out.println(i);  
}
```

# For loop

counter  
variable



A **f** used to do something a given number of times.

```
for (int i = 0;  
    i < 5;  
    i = i + 1) {  
  
    // done 5 times  
    System.out.println(i);  
  
}
```

# For loop

counter  
variable

starting  
value

A **f** used to do something a given number of times.

```
for (int i = 0;  
    i < 5;  
    i = i + 1) {  
  
    // done 5 times  
    System.out.println(i);  
  
}
```

# For loop

counter variable

starting value

A **f**or loop is used to do something a given number of times.

boolean condition

```
for (int i = 0;
    i < 5;
    i = i + 1) {
    // done 5 times
    System.out.println(i);
}
```

# For loop

A **for** loop is used to do something a given number of times.

```
for (int i = 0;  
      i < 5;  
      i = i + 1) {  
    // done 5 times  
    System.out.println(i);  
}
```

counter variable

starting value

boolean condition

increment

# For loop

A for loop is a convenient **short-hand** for a while loop that looks like:

```
int i = 0;
while (i < 5) {
    System.out.println(i);
    i = i + 1;
}
```

# For loop

counter  
variable

A **for** loop is a convenient **short-hand** for a while loop that looks like:

```
int i = 0;
```

```
while (i < 5) {
```

```
    System.out.println(i);
```

```
    i = i + 1;
```

```
}
```

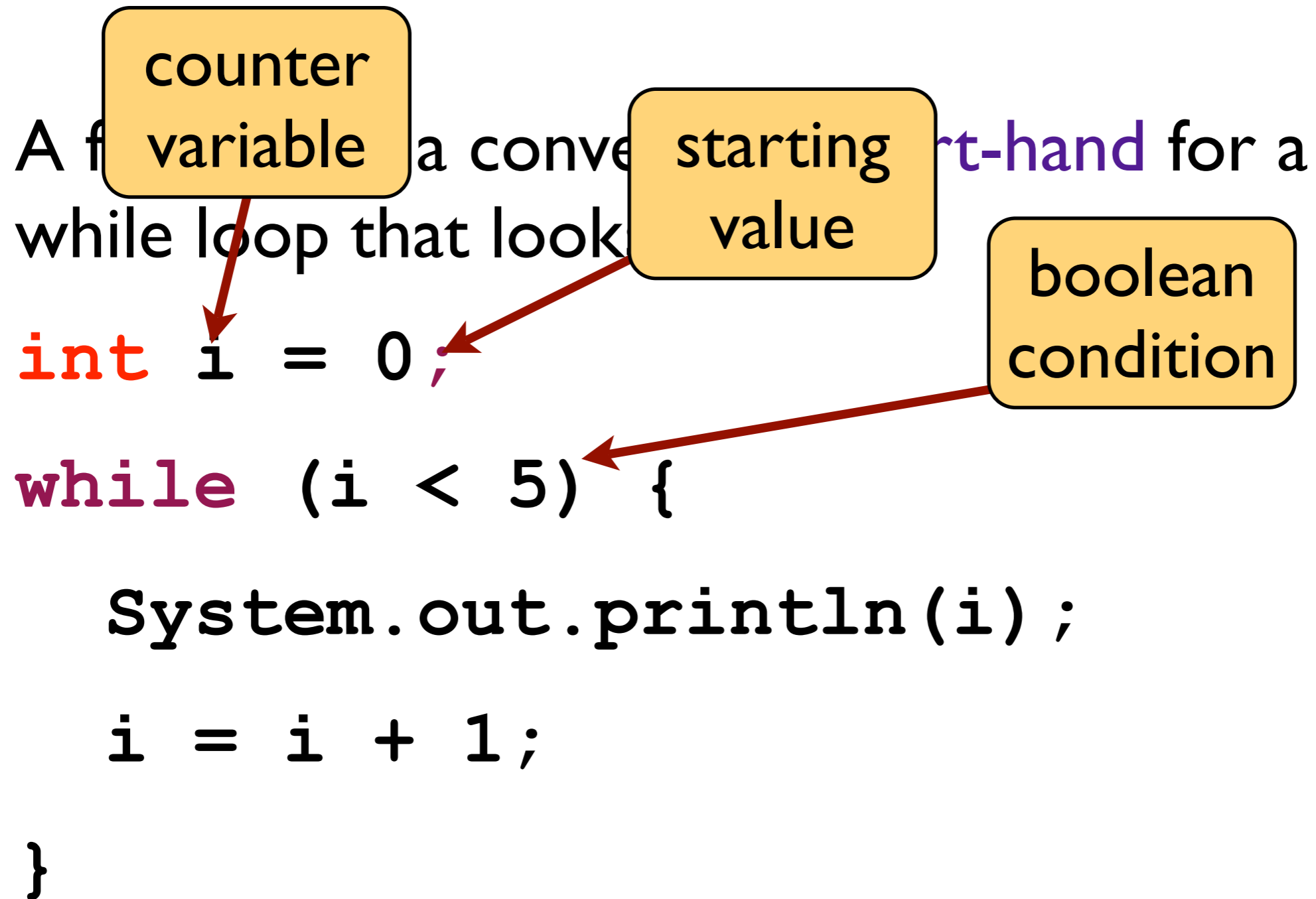
# For loop

A for loop is a convenient way to write a **while** loop that looks like this:

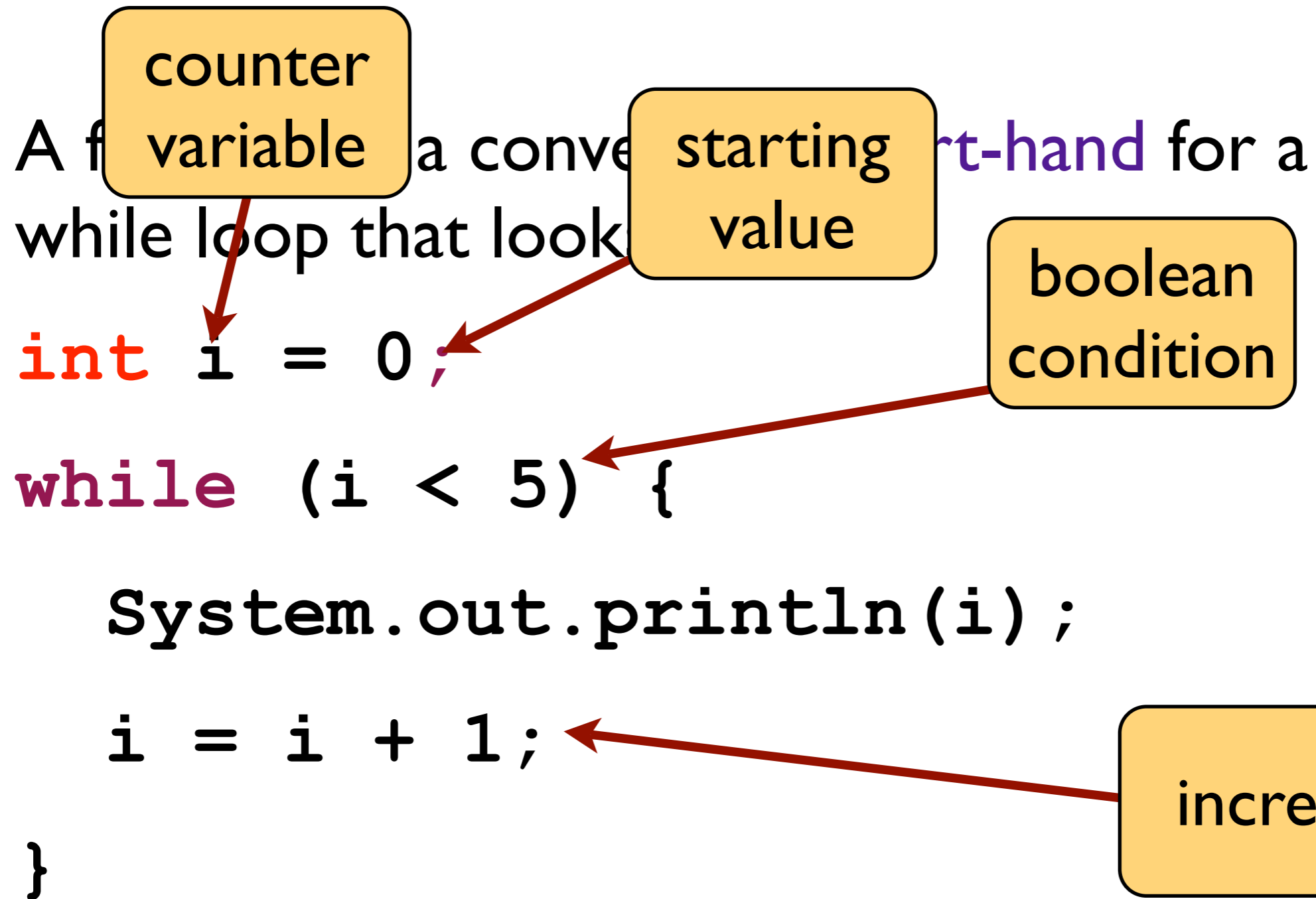
**counter variable**      **starting value**      **end-hand** for a

```
int i = 0;
while (i < 5) {
    System.out.println(i);
    i = i + 1;
}
```

# For loop



# For loop



# For loop

Loop values don't have to be predetermined:

```
for (int i = 0;  
     i < classRoll.size();  
     i++) {  
    System.out.println(classRoll.get(i));  
}
```

Note: `i++` is a short-hand for `i = i + 1`

# For loop

For loops are very flexible:

```
// print every second element  
// backwards from end of list
```

```
for (int i = classRoll.size()-1;  
     i >= 0;  
     i -= 2) {  
    System.out.println(i);  
}
```