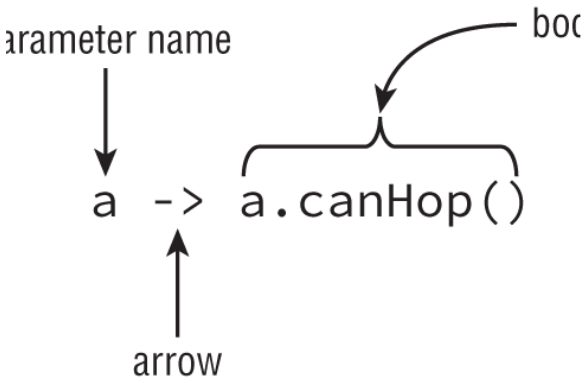
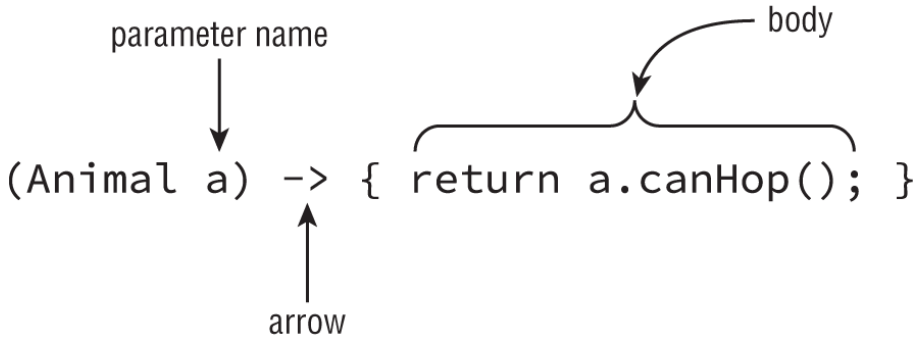
Chapter 6 - Lambdas and Functional interfaces

Lambdas

- *Functional programming* is a way of writing code more declaratively. You specify what you want to do rather than dealing with the state of objects. You focus more on expressions than loops.
- *Deferred execution* means that code is specified now but will run later





Functional Interfaces

- Predicate

```
public interface Predicate<T> {
   boolean test(T t);
}
```

- Consumer

```
void accept(T t)
```

- Supplier

```
T get()
```

- Comparator

```
int compare(T o1, T o2)
```

Functional interface	# parameters	Return type
`Comparator`	Two	`int`
`Consumer`	One	`void`
`Predicate`	One	`boolean`
`Supplier`	None	One (type varies)

Variables in lambda

- Variables can appear in 3 places
  - Parameter list

```
Predicate<String> p = x -> true;
Predicate<String> p = (var x) -> true;
Predicate<String> p = (String x) -> true;
```

  - local variables inside lambda

```
(a, b) -> { int c = 0; return 5;}
```

  - Variable referenced from outside the body

```
public class Crow {
   private String color;
   public void caw(String name) {
      String volume = "loudly";
      Consumer<String> consumer = s ->
            System.out.println(name + " says "
                  + volume + " that she is " + color);
   }
}
```

    Method parameters and local variables are allowed to be referenced if they are *effectively final*

Calling APIs with Lambda

- removeIf

```
bunnies.removeIf(s -> s.charAt(0) != 'h');
```

- sort

```
bunnies.sort((b1, b2) -> b1.compareTo(b2));
```

- foreach

```
bunnies.forEach(b -> System.out.println(b));
```

-