

Lambdas



# Lambdas in Java

- ▶ Lambda expression express instances of *functional interface*, an interface with a single abstract method.
- ▶ Lambda expressions provide the following:
  - ▶ Treat functionality as a method argument or code as data
  - ▶ Allow us to create a function without belonging to any class
  - ▶ An expression that can be passed around as if it were an object and executed on demand
  - ▶ Simpler than anonymous classes with a single method.  
However Lambdas do not have state but anonymous classes may

# Lambda Expression Syntax



```
() -> System.out.println("Zero parameter lambda");
```



```
(p) -> System.out.println("One parameter: " + p);  
//() is optional for one parameter  
//if the type can be inferred from the context  
p -> System.out.println("One parameter: " + p);
```



```
(p, q) -> System.out.println("Multiple parameters: "  
    + p + q);
```

- ▶ See example: [SimpleLambdas.java](#) in class repo



## Method References (1)

- ▶ Consider the following typical use of a lambda.

```
public interface MyPrinter{  
    public void print(String s);  
}  
MyPrinter myPrinter = s -> System.out.println(s);
```

- ▶ The lambda can be replaced by a reference to the `println` method as it simply forwards the string parameter to the `println` method. The double-colon tells the compiler that what follows is a reference to a method.

```
MyPrinter myPrinter = System.out::println;
```

- ▶ Following types of method references may be used:
  - ▶ Static method
  - ▶ Parameter method
  - ▶ Instance method
  - ▶ Constructor

## Method References (2)

- ▶ Static. `ClassName::StaticMethodName`
- ▶ Parameter method

```
public interface Finder {  
    public int find(String s1, String s2);  
}  
Finder finder = String::indexOf;  
//Finder finder = (s1, s2) -> s1.indexOf(s2);
```

- ▶ Instance method

```
public interface Deserializer {  
    public int deserialize(String v1);  
}  
public class StringConverter {  
    public int convertToInt(String v1){  
        return Integer.valueOf(v1);  
    }  
}  
StringConverter stringConverter = new StringConverter();  
Deserializer des = stringConverter::convertToInt;
```

- ▶ Constructors: use the class name followed by `::new`

```
public interface Factory {  
    public String create(char[] val); // matches String  
    constructor  
}  
Factory factory = String::new;  
//Factory factory = chars -> new String(chars);
```