Anonymous Functions

Lambda Expressions

- Lambda Expressions represent *anonymous functions*
  - A function with no name
- It has a formal list of parameters and a body (which can be an expression or a block)
- Consider the function `f(x) = x + 1`. This can be written as a lambda expression:

  ```
  (int x) -> x + 1
  ```
- Java can infer the type, and there can be simplified notation:

  ```
  (x) -> x + 1
  ```

  ```
  x -> x + 1
  ```

  though you are expected to use the first method if you use lambda expressions

But... But... WHY?

- A language that allows lambda expressions considers functions to be *first order objects*
    - functions are things too!
- Languages that treat functions this way are *Cool Things*™
- We will examine one use of these in a few slides

Functions as Things

Java defines many functional objects (in `java.util.function`):

- `IntFunction<R>`
  - A function with one `int` parameter, returns type R
  - Has a method `apply(int)`
- `DoubleFunction<R>`
  - A function with one `double` parameter, returns type R
  - Has a method `apply(double)`
- `Function<T, R>`
  - A function with one T parameter, returns type R
  - Has a method `apply(T)`
- `Function<T, U, R>`
  - A function with one T and one U parameter, returns type R
  - Has a method `apply(T, U)`
- `Predicate<T>`
  - A boolean-valued function with an argument of type T
  - Has a method `test(T)`

Autoboxing

- Java does not allow auto-boxing on the parameters types of the lambda expressions
- `IntFunction` requires an `int` and will not accept an `Integer`
  - This does not apply to the return values

Coding Examples

```
IntFunction<Integer> timesTwo = (int x) -> x * 2;
System.out.println(timesTwo.apply(5));

Function<String, Integer> doubleTheLength;
doubleTheLength = (String s) -> 2 * s.length();
System.out.println(doubleTheLength.apply("Bob"));

BiFunction<Integer, Integer, Integer> doubleThenAdd;
doubleThenAdd = (Integer a, Integer b) -> 2 * a + b;
System.out.println(doubleThenAdd.apply(20,12));
```

Functions as Return Types

- Lambdas can be the return type of a method
  - This means you can build lambda expressions in methods and return them.

```
public static IntFunction<Integer> add7xN(int n)
{
  return (int x) -> x + 7 * n;
}

public static void main(String[] args)
{
  System.out.println(add7xN(2).apply(10));
  System.out.println(add7xN(3).apply(10));
}
```

Sorting with Lambda Expressions

- Java 8 introduced a default method named `sort` in the `List` interface that takes one argument: a comparator function!
- The elements in the object are sorted according to the comparator passed as the argument.
- General syntax:

  `aList.sort(comparator)`

  `Collectios.sort(nums, comparator)`
- Example:

```
List<Integer> nums = Arrays.asList(2, 9, 4, 6, 1, 8, 5);
System.out.println(nums);
nums.sort((Integer x, Integer y) -> Integer.compare(x, y));
System.out.println(nums);
```