Functions

Monday, March 12, 2018 11:51 AM

Abstraction of functionality of methods

Functions

Vs. methods

- Similar to mathematics (domain -> codomain, image)

Pure function

- Deterministic
- No side effects (printing, changing values, exceptions)
- Applied to classes: immutable (e.g. String)
- Functional Programming (FP) or functional-style programming within an OO language
- Bugs are usually due to side effects
- OO = encapsulate moving parts, FP = minimise moving parts

Partial function

- Not all elements in domain mapped

Function interface

- `Function<T, R>`
- `R apply(T t) // abstract method`
- `applyList // map`
- Functions can be composed at run time, dynamically

Lambda Expression

```
applyList(list, (Integer x) -> { return x * x; });
applyList(list, x -> { return x * x; });
applyList(list, x -> x * x);
```

- Use () for no argument
- Can be applied to variables
- Lambda expressions are actually anonymous classes
    - Can only access final/ effectively final local variables
    - Members of enclosing class

Method Reference

- `Function<Integer, Integer> f = Math::abs`

Composing Functions

- `andThen, compose`
- Widening type conversion in generics
    - Producer extends; consumer super (PECS)
    - Can take in super R -> can take in R
    - Can produce extends T -> can produce T

Other Interfaces

`Predicate<T>` with a `boolean test(T t)` method

`Supplier<T>` with a `T get()` method

`Consumer<T>` with a `void accept(T t)` method

`BiFunction<T,U,R>` with a `R apply(T t, U u)` method

- printer/ randInt are impure

Curried Functions

- Arity = number of arguments/ operands
- Higher-order functions
  - Functions with multiple arguments can be built with unary functions (currying)
  - A sequence of curried functions
- Useful when not all argument are available first (partial application of function)
- One of arguments does not change often/ is expensive to compute

Exercises

1. Pure functions
   - f throws an exception
   - g prints a statement
   - h changes the random generator
     - Takes a random seed from the entropy of the system
     - Output not deterministic
4. Curried functions
   - `Function<T, Function<T, Function<T, R>>>`
   - `exp.apply(x).apply(y).apply(z);`
5. LambdaList
   - `T… varargs`
     - Represents passing in an array
     - Results in warning because of possible ClassCastException
     - Arrays all become Object[] during compilation
     - `@SafeVarargs`