CSE214 Data Structures
Lambda

YoungMin Kwon

SUNY Korea

Lambda

- Lambda expression
  - Provides a body for a Single Abstract Method
  - default method: an interface method with a body

```
public class AnonymousClass2 {
    public interface Adder {
        public  int add(int a, int b);  //Single Abstract Method
        default int sub(int a, int b) { //default implementation
            return add(a, -b);
        }
    }

    public static void main(String[] args) {
        Adder adder = (a, b) -> a + b; //Lambda expression
        System.out.println("adder.add(3, 2): " + adder.add(3, 2));
        System.out.println("adder.sub(3, 2): " + adder.sub(3, 2));
    }
}
```

SUNY Korea

```
public class LambdaExercise {
    //EPS is a small number
    public static final double EPS = 1e-10;

    //function
    public interface Fun<T, R> {
        //Single Abstract Method
        public R apply(T a);
    }

    //recursive function
    public interface Rec<T, R> {
        //Single Abstract Method
        public R fun(Rec<T, R> self, T a);
        public default R apply(T a) {
            return fun(this, a);
        }
    }

    //recursive function with 2 parameters
    public interface Rec2<T1, T2, R> {
        //Single Abstract Method
        public R fun(Rec2<T1, T2, R> self, T1 a, T2 b);
        public default R apply(T1 a, T2 b) {
            return fun(this, a, b);
        }
    }
```

this in lambda expr. references the this of the surrounding env.

SUNY Korea

Factorial and GCD

```
//Factorial
public static Rec<Integer, Integer> fact =
    (self, a) -> a <= 1 ? a
               : a * self.apply(a - 1);

//GCD
public static Rec2<Integer, Integer, Integer> gcd =
    (self, a, b) -> a > b ? self.apply(a - b, b)
                  : b > a ? self.apply(b - a, a)
                  : a;
```

SUNY Korea

Derivative, Scale, Add

```
//derivative
//TODO: deriv(f) returns a derivative of f
//      i.e. deriv(f)(x) = (f(x + EPS) - f(x)) / EPS
public static Fun<Fun<Double,Double>,       /*e.g. f  */
                  Fun<Double,Double>>       /*e.g. f' */
derivative =

protected static void test_deriv() {
    Fun<Double, Double> sin = x -> Math.sin(x);
    Fun<Double, Double> cos = x -> Math.cos(x);

    //TODO: sinDeriv is the derivative of sin
    //      using deriv and sin implement sinDeriv
    Fun<Double, Double> sinDeriv =

    for(double x = 0; x < 3.14; x += 0.1) {
        double a = cos.apply(x);
        double b = sinDeriv.apply(x);
        onFalseThrow(equ(a, b));
    }
}
```

SUNY Korea

```
//scale
//TODO: scale(s) takes a function f and returns a function scaled by s
//      i.e. scale(5)(f)(x) = 5*f(x)
public static Fun<Double,                       /*e.g. scale factor 5*/
                  Fun<Fun<Double, Double>,  /*e.g. function f*/
                      Fun<Double, Double>>> /*e.g. 5*f*/
scale =

protected static void test_scale() {
    Fun<Double, Double> sin = x -> Math.sin(x);

    //TODO: scale5 is a function that takes f and scale it by 5.0
    //      i.e. scale5(f)(x) = 5 * f(x).
    //      using scale implement scale5
    Fun<Fun<Double,Double>, Fun<Double,Double>> scale5 =

    //TODO: sin5(x) = 5 * sin(x)
    //      using scale5 and sin implement sin5
    Fun<Double, Double> sin5 =

    for(double x = 0; x < 3.14; x += 0.1) {
        double a = 5*sin.apply(x);
        double b = sin5.apply(x);
        onFalseThrow(equ(a, b));
    }
}
```

SUNY Korea

```
//add
//TODO: add(f) takes a function g and returns a function f + g
//      i.e. add(f)(g)(x) = f(x) + g(x)
public static Fun<Fun<Double,Double>,        /*e.g. f    */
                  Fun<Fun<Double, Double>,   /*e.g. g    */
                      Fun<Double, Double>>>  /*e.g. f + g*/
add =

protected static void test_add() {
    Fun<Double, Double> sin = x -> Math.sin(x);
    Fun<Double, Double> cos = x -> Math.cos(x);
    //TODO: sinPlus is a function that takes f and add sin to it
    //      i.e. sinPlus(f)(x) = f(x) + sin(x).
    //      using add and sin implement sinPlus
    Fun<Fun<Double,Double>, Fun<Double,Double>> sinPlus =

    //TODO: sinPlusCos(x) = sin(x) + cos(x)
    //      using sinPlus and cos implement sinPlusCos
    Fun<Double, Double> sinPlusCos =

    for(double x = 0; x < 3.14; x += 0.1) {
        double a = sin.apply(x) + cos.apply(x);
        double b = sinPlusCos.apply(x);
        onFalseThrow(equ(a, b));
    }
}
```

SUNY Korea

```
protected static void test_linearity() {
    Fun<Double, Double> sin = x -> Math.sin(x);
    Fun<Double, Double> cos = x -> Math.cos(x);

    //TODO: sin3(x) = 3*sin(x).
    //      using scale, sin implement sin3
    Fun<Double, Double> sin3 =

    //TODO: cos5(x) = 5*cos(x).
    //      using scale, cos implement cos5
    Fun<Double, Double> cos5 =

    //TODO: com(x) = 3*sin(x) + 5*cos(x).
    //      using add, sin3 and cos5 implement com
    Fun<Double, Double> com =

    for(double x = 0; x < 3.14; x += 0.1) {
        double a = 3.*sin.apply(x) + 5.*cos.apply(x);
        double b = com.apply(x);
        onFalseThrow(equ(a, b));
    }
}
```

SUNY Korea

```
    protected static boolean equ(double a, double b) {
        return Math.abs(a - b) < 1e-5;
    }

    protected static void onFalseThrow(boolean b) {
        if(!b)
            throw new RuntimeException("Error: unexpected");
    }

    public static void main(String[] args) {
        test_deriv();
        test_scale();
        test_add();
        test_linearity();
        System.out.println("Success!");
    }
}
```

SUNY Korea