Part 1 -Pizza “treats”/traits

- The goal is to define set of traits allowing to compose variety of pizzas
- Each trait adds a some topping that has a name and a price
- The code below should compile and work as expected:

```
  val myPizza = new ThinDough with TomatoSauce with Mozarella with Ham
  println( myPizza.name + myPizza.price )

  val yourPizza = new ThinDough with TomatoSauce with Mozarella with Mushroms
  println( yourPizza.name + yourPizza.price )
/* expected result
Ham  Mozarella  Tomato Sauce  on thin dough 14.0
Mozarella  Mozarella  Tomato Sauce  on thin dough 14.5
*/
```

name and price are methods!

Part 2 - generics

The goal is to write a **covariant** (therefore the name) container for a **Pair of objects of the same type**. Functionality as in example code.

```
class A{
    override def toString: String = "A"
  }
  class B( val x: Int) extends A{
    override def toString: String = "B:"+x.toString
  }
  class C( x: Int) extends B(x){
    override def toString: String = "C:"+x.toString
  }

  val a: TwistedMonoPair[A] = TwistedMonoPair[A](new B(7), new A)
  println(a(0))
  println(a(1))
  println(a)
  val b: TwistedMonoPair[A] = TwistedMonoPair[B](new B(9), new B(77)) // covariantnes
  println(b)
  // val c: TwistedMonoPair[A] = new TwistedMonoPair[B](new B(9), new A) // should not compile because of
second argument of c’tor

  val d1 = b.replace(0)(new A) // conversion to TwistedMonoPair[A] and replacement of the first el. in the pair
  println(d1)
  val tA : TwistedMonoPair[A] = d1
  //val tB : TwistedMonoPair[B] = d1// can not compile
  val d2 = b.replace(1)(new A) // as above, but replaced is the second el
  println(d2)
```