// Traits in Scala
// Gustav Cedersjö

```
// Outline

// Part I:   Interfaces
// Part II:  Adding functionality
// Part III: Stackable modifications
```

```
// Part I: Interfaces

// Interface for a queue of integers
trait IntQueue {
  def get(): Int
  def put(x: Int)
}
```

```
// A simple implementation of IntQueue
import scala.collection.mutable.ArrayBuffer
class BasicIntQueue extends IntQueue {
  private val buf = new ArrayBuffer[Int]()
  def get() = buf.remove(0)
  def put(x: Int) { buf += x }
}
```

```
// Demo time!
val q1 = new BasicIntQueue
q1.put(1)
q1.put(2)
q1.get()
q1.get()
```

```
// Part II: Adding functionality

// We define a trait with new methods
trait MultiPut extends IntQueue {
  def putAll(xs: Int*) {
    for(x <- xs) { put(x) }
  }
}
```

```
// How can we use the new trait?

// Make a new class and mix in the new trait
class MultiPutIntQueue extends
  BasicIntQueue with MultiPut
val q2 = new MultiPutIntQueue

// ...or mix in the new trait on construction
val q3 = new BasicIntQueue with MultiPut
```

```
// Part III: Stackable modifications

// Double the value of the ints
trait Doubling extends IntQueue {
  abstract override def put(x: Int) {
    super.put(x+x)
  }
}
```

```
// Demo time!
val q4 = new BasicIntQueue with Doubling
q4.put(1)
q4.put(2)
q4.get()
q4.get()
```

```
// What about more modifications?
trait Incrementing extends IntQueue {
  abstract override def put(x: Int) {
    super.put(x+1)
  }
}

// We stack to modifications to the queue
val q5 = new BasicIntQueue with
  Doubling with Incrementing

// In which order are the traits used?
q5.put(1)
q5.put(2)
q5.get()
q5.get()
```

```
// Of course we can also mix in
// the MultiPut trait
val q6 = new BasicIntQueue
  with Doubling with Incrementing with MultiPut
q6.putAll(1,2,3)
q6.get()
q6.get()
q6.get()
```

```
// Exercise...
```