TOUR OF SCALA

GENERIC CLASSES

Generic classes are classes which take a type as a parameter. They are particularly useful for collection classes.

Defining a generic class

Generic classes take a type as a parameter within square brackets `[]`. One convention is to use the letter `A` as type parameter identifier, though any parameter name may be used.

```
class Stack[A] {
  private var elements: List[A] = Nil
  def push(x: A): Unit =
    elements = x :: elements
  def peek: A = elements.head
  def pop(): A = {
    val currentTop = peek
    elements = elements.tail
    currentTop
  }
}
```

This implementation of a `Stack` class takes any type `A` as a parameter. This means the underlying list, `var elements: List[A] = Nil`, can only store elements of type `A`. The procedure `def push` only accepts objects of type `A` (note: `elements = x :: elements` reassigns `elements` to a new list created by prepending `x` to the current `elements`).

`Nil` here is an empty `List` and is not to be confused with `null`.

Usage

To use a generic class, put the type in the square brackets in place of `A`.

```
val stack = new Stack[Int]
stack.push(1)
stack.push(2)
println(stack.pop)  // prints 2
println(stack.pop)  // prints 1
```

The instance `stack` can only take Ints. However, if the type argument had subtypes, those could be passed in:

```
class Fruit
class Apple extends Fruit
class Banana extends Fruit

val stack = new Stack[Fruit]
val apple = new Apple
val banana = new Banana

stack.push(apple)
stack.push(banana)
```

Class `Apple` and `Banana` both extend `Fruit` so we can push instances `apple` and `banana` onto the stack of `Fruit`.

*Note: subtyping of generic types is *invariant*. This means that if we have a stack of characters of type `Stack[Char]` then it cannot be used as an integer stack of type `Stack[Int]`. This would be unsound because it would enable us to enter true integers into the character stack. To conclude, `Stack[A]` is only a subtype of `Stack[B]` if and only if `B = A`. Since this can be quite restrictive, Scala offers a type parameter annotation mechanism to control the subtyping behavior of generic types.*

← previous next →

Contributors to this page:

ckipp01 Nexus01 mlachkar SethTisue ashawley manishpatelUK heathermiller