

Lecture 30: Exceptions

CS 51G
Spring 2018
Kim Bruce

Announcements

- Exercise 15.5.3
- Simon lab Friday
- Courses for next year
 - CS 52 & 55 vs 54
 - For most, CS 54 should be best choice

Strings are Lists!

- ... but immutable. Can get individual elements via `at(i)`
- Can also iterate through characters in string
 - Does a string represent a number?

```
method isInteger(word:String) -> Boolean {  
  for(word) do {letter: String ->  
    if (("0" > letter) || (letter > "9")) then {  
      return false  
    }  
  }  
  true  
}
```

Strings are ordered

- `<`, `<=`, `>`, `>=`, `==`, and `!=` work as expected (alphabetical order).
- `ord` gives numeric code of first letter of string

Exceptions

Exceptional Conditions

- What do you do when something goes wrong?
 - Try to handle nicely within code. Sometimes that's not possible and have to give up current computation.
- Language construct to catch run-time errors

```
try {  
    stuff to try  
} catch{ ex: SomeExceptionType ->  
    // stuff to do if exception occurs  
}
```

Example from Objectdraw

```
def ColorOutOfRange: prelude.ExceptionKind is public =  
  prelude.ProgrammingError.refine "ColorOutOfRange"
```

```
def colorGen is public = object {  
  class r (r': Number) g (g': Number) b (b': Number) -> Color {  
    // Creates a color with rgb coordinates r', g', and b'  
    if ((r' < 0) || (r' > 255)) then {  
      ColorOutOfRange.raise "red index {r'} out of bounds 0..255"  
    }  
  
    if ((g' < 0) || (g' > 255)) then {  
      ColorOutOfRange.raise "green index {g'} out of bounds 0..255"  
    }  
  
    if ((b' < 0) || (b' > 255)) then {  
      ColorOutOfRange.raise "blue index {b'} out of bounds 0..255"  
    }  
  }  
}
```

ColorGen continued

```
def red:Number is public = r'.truncated
def green:Number is public = g'.truncated
def blue:Number is public = b'.truncated
```

```
method == (c: Color) -> Boolean {
  (red == c.red) && (green == c.green) && (blue == c.blue)
}
```

```
method asString -> String {
  "color w/ rgb({red}, {green}, {blue})"
}
}
```

```
method random -> Color {
  // Produce a random color.
  r (randomIntFrom (0) to (255))
  g (randomIntFrom (0) to (255))
  b (randomIntFrom (0) to (255))
}
```

ColorGen continued

```
def white:Color is public = r (255) g (255) b (255)
def black:Color is public = r (0) g (0) b (0)
def green:Color is public = r (0) g (255) b (0)
def red:Color is public = r (255) g (0) b (0)
def gray:Color is public = r (60) g (60) b (60)
def blue:Color is public = r (0) g (0) b (255)
def cyan:Color is public = r (0) g (255) b (255)
def magenta:Color is public = r (255) g (0) b (255)
def yellow:Color is public = r (255) g (255) b (0)
def neutral:Color is public = r (220) g (220) b (220)
}
```

Using the Exception

```
method changeColor -> Done {
  var newColor: Color
  try {
    newColor := color.r(redField.number)
                      g(greenField.number)
                      b(blueField.number)
  } catch {
    ex: ColorOutOfRange ->
      print "Enter values between 0 and 255 for colors"
      newColor := black
  }
  background.color := newColor
}
```

Using Parameter ex

- Here are some of its methods:
 - `exceptionKind` → `exceptionKind` answers the `exceptionKind` of this exception.
 - `message` → `String` the message that was provided when this exaction was raised.
 - `data` → `Object` answers the data object that was associated with this exception when it was raised, if there was one. Otherwise, answers the string “no data”.
 - `lineNumber` → `Number` the source code line number of the request of raise that created this exception.
 - `backtrace` → `List<String>` a list of strings describing the call stack at the time that this exception was raised. `backtrace.first` is the initial execution environment; `backtrace.last` is the context that raised the exception.

Another Example

```
def myList:List[[Number]] = list[[Number]] [5,7,9]
var index := 1
try {
  while {index < 7} do {
    print(myList.at(index))
    index := index + 1
  }
} catch {ex: BoundsError ->
  print "went too far!"
  print ("on line {ex.lineNumber} of {ex.moduleName}, {ex.message}")
  print "\n\nBacktrace: {ex.backtrace}"
}
```

Questions?