Pattern Matching

Solution 2: Functional Decomposition with Pattern Matching

- Observation: The sole purpose of test and accessor functions is to reverse the construction process (what subclass was used, what were the arguments of the constructor)
- This situation is so common that many functional languages, Scala included, automate it

Case Classes

- Scala supports functional decomposition through case classes
- A case class definition is similar to a normal class definition, except that it is preceded by the modifier case
- For example:

```
trait Expression:
case class Number(n: Int) extends Expression
case class Sum(e1: Expression, e2: Expression) extends Expression
```

- Like before, this defines a trait Expression and two concrete subclasses Number and Sum

Pattern Matching

- Pattern matching is a generalization of **switch** from C/Java to class hierarchies
- It's expressed in Scala using the keyword ***match***
- Example:

```
def evaluate(e: Expression): Int = e match
  case Number(n) => n
  case Sum(e1, e2) => evaluate(e1) + evaluate(e2)
```

- What you see here is a pattern matching that at the same time identifies a case (Number or Sum) and names the elements in that case

Match Syntax

Rules:

- ***Match*** is preceded by a selector expression and is followed by a sequence of ***cases***, ***pattern => expression***
- Each case associates an expression with a pattern
- A *MatchError* exception is thrown if no pattern matches the value of the selector

Forms of Patterns

- Patterns are constructed from:
  - Constructors: Number, Sum
  - Variables: n, e1, e2
  - Wildcards patterns: _
  - Constants: 1, true
  - Type tests: n: Number
- Variables always begin with a lowercase letter
- The same variable name can only appear once in a pattern (Sum(x, x) is not a legal pattern)

- Names of constants begin with a capital letter, with the exception of the reserved words: null, false, true

Evaluating Match Expressions

- An expression of the form ***e match { case p_1 => e_1 … case p_n => e_n }*** matches the value of the selector e with the patterns p_1, … p_n in the order in which they are written
- The whole expression is rewritten to the right-hand side of the first case where the pattern matches the selector e
- References to pattern variables are replaced by the corresponding parts in the selector

What do Patterns Match?

- A constructor pattern C(p_1, … p_n) matches all the values of type C (or a subtype) that have been constructed with arguments matching the patterns p_1, … p_n
- A variable pattern x matches any value and binds the name of the variable to this value; a wildcard pattern matches any value and does not bind any name to that value
- A constant pattern c matches values that are equal to c (in the sense of ==)
- A pattern like n: Number would match any value that is a number and bind it with a name n

Exercise: *Write a function show that uses pattern matching to return the representation of a given expression as a string.*

```
def show(e: Expression): String = e match
    case  Number(n) => n.toString
    case Sum(e1, e2) => s"${show(e1)} + ${show(e2)}"
```