

Scala - XML, combinator parsing, actors

Grzegorz Balcerek

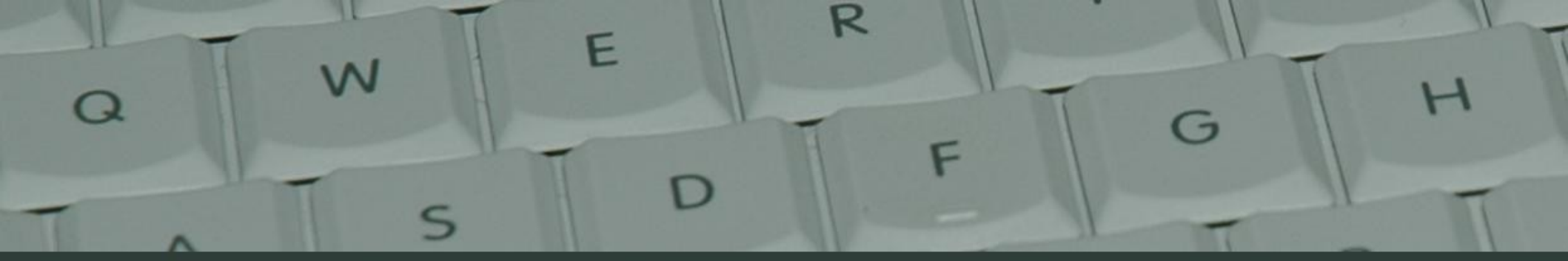
`gbalcerek@echostar.pl`

This work is licensed under a
Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License
(<http://creativecommons.org/licenses/by-nc-sa/3.0/>).



spis treści

- przetwarzanie XML
- parsowanie - combinator parsing
- programowanie współbieżne - actors



XML

XML

budowanie XML

```
scala> val a = <a/>
```

```
a: scala.xml.Elem = <a></a>
```

```
scala> val b = <b>123</b>
```

```
b: scala.xml.Elem = <b>123</b>
```

```
scala> val c = <c>{ 2+5 }</c>
```

```
c: scala.xml.Elem = <c>7</c>
```

```
scala> val d = <d id={ "x"+"y" }>abc</d>
```

```
d: scala.xml.Elem = <d id="xy">abc</d>
```

```
scala> val e = <e>x<f>y</f>z</e>
```

```
e: scala.xml.Elem = <e>x<f>y</f>z</e>
```

XML

budowanie XML

```
scala> val err = <div><span>abc</div>
```

```
<console>:1: error: in XML literal: expected closing tag of  
span
```

```
    val err = <div><span>abc</div>  
                                     ^
```

```
<console>:1: error: start tag was here: span>
```

```
    val err = <div><span>abc</div>  
               ^
```

```
scala>
```

XML

odczyt danych z XML

```
scala> val p = <p id="1"><div>ab<b>17</b></div></p>  
p: scala.xml.Elem = <p id="1"><div>ab<b>17</b></div></p>
```

```
scala> val r = p.text  
r: String = ab17
```

```
scala> val r = p \ "@id"  
r: scala.xml.NodeSeq = 1
```

```
scala> val r = (p \ "@id").text  
r: String = 1
```

```
scala> val r = p \ "div"  
r: scala.xml.NodeSeq = NodeSeq(<div>ab<b>17</b></div>)
```

```
scala> val r = p \ "b"  
r: scala.xml.NodeSeq = NodeSeq()
```

XML

odczyt danych z XML

```
scala> val p = <p id="1"><div>ab<b>17</b></div></p>  
p: scala.xml.Elem = <p id="1"><div>ab<b>17</b></div></p>
```

```
scala> val r = p \\ "b"  
r: scala.xml.NodeSeq = NodeSeq(<b>17</b>)
```

```
scala> val r = (p \\ "b").text  
r: String = 17
```

```
scala> val r = (p \\ "b").text.toInt  
r: Int = 17
```

XML

zapis/odczyt pliku XML

```
scala> val content = <body>hello</body>  
content: scala.xml.Elem = <body>hello</body>
```

```
scala> XML.save("example.xml", content, "utf-8", true)
```

```
scala>
```

```
<?xml version='1.0' encoding='utf-8'?>  
<body>hello</body>
```

```
scala> val content = XML.load("example.xml")  
content: scala.xml.Elem = <body>hello</body>
```

XML

pattern matching

```
scala> def examine(n: Node): String = n match {  
  | case <b>{c}</b>           => "1)<b>:" + c  
  | case <p>{c : Text}</p>    => "2)<p> with text:" + c  
  | case Elem(null,e,_,_,c) => "3)Elem " + e + ":" + c  
  | case _                   => "4)Something else"  
  | }  
examine: (n: scala.xml.Node)String
```

```
scala> val r = examine(<b>123</b>)  
r: String = 1)<b>:123
```

```
scala> val r = examine(<p>123</p>)  
r: String = 2)<p> with text:123
```

```
scala> val r = examine(<p><b>123</b></p>)  
r: String = 3)Elem p:<b>123</b>
```

```
scala> val r = examine(<a>123<b/></a>)  
r: String = 4)Something else
```

XML

pattern matching

```
scala> val list = List(2,5,7,9)
list: List[Int] = List(2, 5, 7, 9)
```

```
scala> val p = <p>{ for (e <- list) yield <b>{e}</b> }</p>
p: scala.xml.Elem = <p><b>2</b><b>5</b><b>7</b><b>9</b></p>
```

```
scala> var r = for (<b>{e}</b> <- p\"b\") yield e.text.toInt
r: scala.collection.immutable.Seq[Int] = List(2, 5, 7, 9)
```

```
scala> r = for (Elem(_,_,_,_,e) <- p\"b\") yield e.text.toInt
r: scala.collection.immutable.Seq[Int] = List(2, 5, 7, 9)
```

XML

przykład: Sudoku.scala

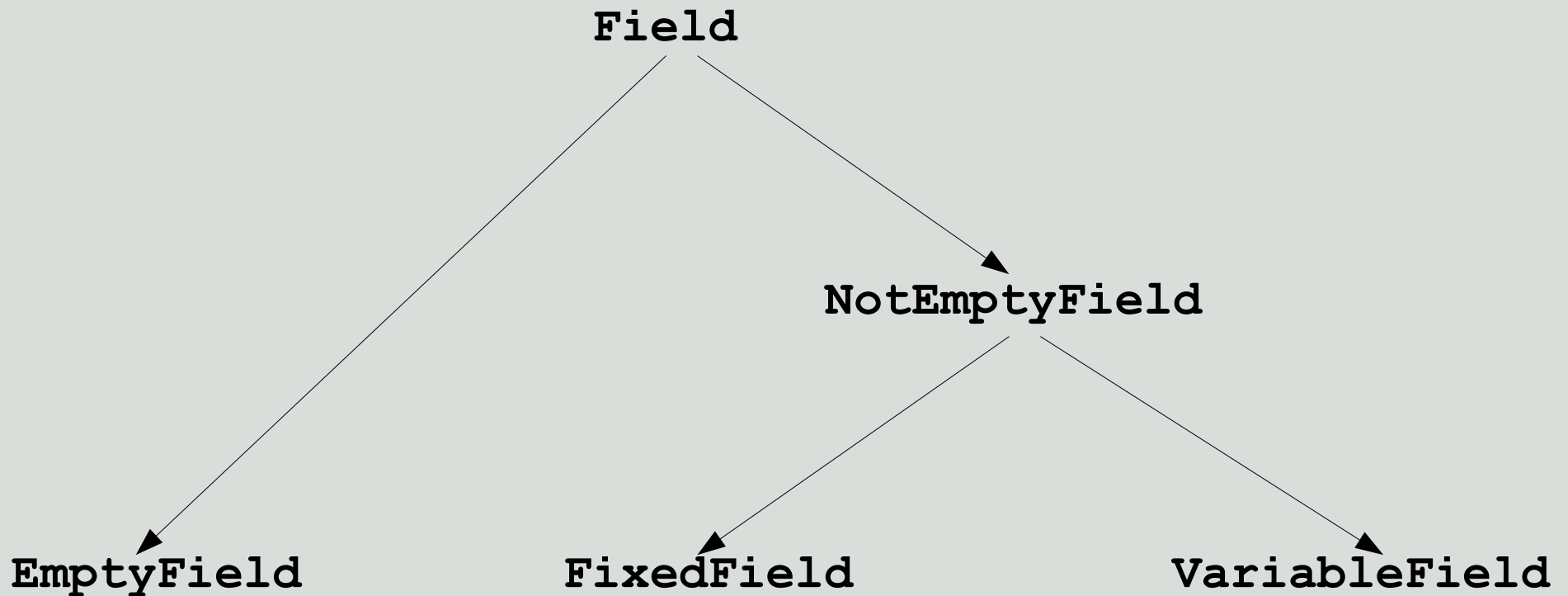


A screenshot of a window titled "Sudoku" with a menu bar containing "File" and "Sudoku". The window displays a 9x9 grid of numbers. The numbers are arranged as follows:

4		6				1	5	7
1	5	3	4	6	7	9	8	2
			1	2	5	6	3	4
2	1	5	7	9	3	4	6	8
3	6	4	2	5	8	7	1	9
8	9	7	6	1	4	5	2	3
9	7	8	5	3	6	2	4	1
6	4	2	8	7	1	3	9	5
5	3	1	9	4	2	8	7	6

XML

Sudoku: struktury danych



```
val board: Array[Field] = Array.ofDim(81)
```

XML

Sudoku: zapis do pliku xml

Array[Field]



```
<?xml version='1.0' encoding='utf-8'?>
<board>
<variable num="0">4</variable>
<empty num="1"></empty>
<fixed num="2">6</fixed>
(...)
<variable num="80">6</variable></board>
```

XML

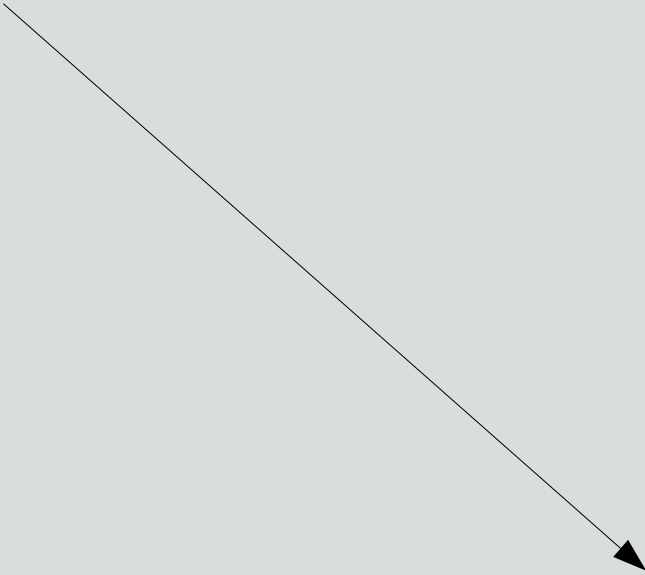
Sudoku: zapis do pliku xml

```
override def save(file: File, board: Array[Field]) {
  val content =
    <board>{
      for(j <- 0 to 80; field = board(j)) yield
        Text("\n") :+
        (field match {
          case EmptyField =>
            <empty num={j.toString}/>
          case FixedField(value) =>
            <fixed num={j.toString}>{value}</fixed>
          case VariableField(value) =>
            <variable num={j.toString}>{value}</variable>
        })
    }</board>
  XML.save(file.getPath, content, "utf-8", true)
}
```

XML

Sudoku: odczyt z pliku xml

```
<?xml version='1.0' encoding='utf-8'?>  
<board>  
<variable num="0">4</variable>  
<empty num="1"></empty>  
<fixed num="2">6</fixed>  
(...)  
<variable num="80">6</variable></board>
```



Array[Field]

XML

Sudoku: odczyt z pliku xml - wersja 1

```
override def load(file: File, board: Array[Field]) {  
  val content = XML.load(file.getPath)  
  for (j <- 0 to 80) board(j) = EmptyField  
  for (field <- content \ "fixed" ) {  
    val num = (field \ "@num").text.toInt  
    val value = field.text.toInt  
    board(num) = FixedField(value)  
  }  
  for (field <- content \ "variable" ) {  
    val num = (field \ "@num").text.toInt  
    val value = field.text.toInt  
    board(num) = VariableField(value)  
  }  
}
```

XML

Sudoku: odczyt z pliku xml - wersja 2

```
override def load(file: File, board: Array[Field]) {
  val source = scala.io.Source.fromFile(file)
  val reader = new XMLEventReader(source)
  var attributes: MetaData = null
  var text: String = null
  for (event <- reader) event match {
    case EvElemStart(null, "board", _, _) =>
      for (j <- 0 to 80) board(j) = EmptyField
    case EvElemStart(null, _, attr, TopScope) =>
      attributes = attr
    case EvText(t) => text = t
    case EvElemEnd(null, name)
      if (name == "variable" ||
          name == "fixed") =>
      setField(board, name, attributes, text)
    case _ =>
  }
}
```

XML

Sudoku: odczyt z pliku xml - wersja 2

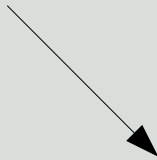
```
private def setField(board: Array[Field],
    name: String, attributes: MetaData,
    text: String) {
    val num = findNum(attributes)
    val value = text.toInt
    board(num) = name match {
        case "fixed" => FixedField(value)
        case "variable" => VariableField(value)
    }
}

def findNum(attributes: MetaData) =
    attributes find (_.key == "num") map
        (_.value.head.text.toInt) getOrElse(0)
```

XML

Sudoku: przekształcenie na html

```
<?xml version='1.0' encoding='utf-8'?>
<board>
<variable num="0">4</variable>
<empty num="1"></empty>
<fixed num="2">6</fixed>
(...)
<variable num="80">6</variable></board>
```



```
<?xml version='1.0' encoding='utf-8'?>
<html>
(...)
  <body>
    <h1>Sudoku</h1>
    <table>
      <tr><td>4</td><td></td><td><strong>6</strong> (...)
```

Sudoku

4	6			1	5	7		
1	5	3	4	6	7	9	8	2
			1	2	5	6	3	4
2	1	5	7	9	3	4	6	8
3	6	4	2	5	8	7	1	9
8	9	7	6	1	4	5	2	3
9	7	8	5	3	6	2	4	1
6	4	2	8	7	1	3	9	5
5	3	1	9	4	2	8	7	6

XML

Sudoku: przekształcenie na html

```
object SudokuTransformer extends BasicTransformer {
  override def transform(n: Node) = n match {
    case <board>{content @ _*}</board> =>
<html>
  <head>
    <title>Sudoku</title>
    <style>
      table {{ border-collapse: collapse }}
      td {{ border: 1px solid }}
    </style>
  </head>
  <body>
    <h1>Sudoku</h1>
    <table>
      { transformFields(content filter (isSudokuField(_))) }
    </table>
  </body>
</html>
```

XML

Sudoku: przekształcenie na html

```
case <empty/> =>
  <td/>
case <fixed>{value}</fixed> =>
  <td><strong>{value}</strong></td>
case <variable>{value}</variable> =>
  <td>{value}</td>
case _ => n
}
private def isSudokuField(n: Node) = n match {
  case <empty/> => true
  case <fixed>{ _ }</fixed> => true
  case <variable>{ _ }</variable> => true
  case _ => false }
private def transformFields(fields: NodeSeq): NodeSeq =
  if (fields.isEmpty) NodeSeq.Empty
  else <tr>{transform(fields.take(9))}</tr> ++
    transformFields(fields.drop(9))
}
```



combinator parsing

combinator parsing

przykład: Sudoku



A screenshot of a window titled "Sudoku" showing a 9x9 grid. The grid contains numbers and empty cells. The numbers are arranged as follows:

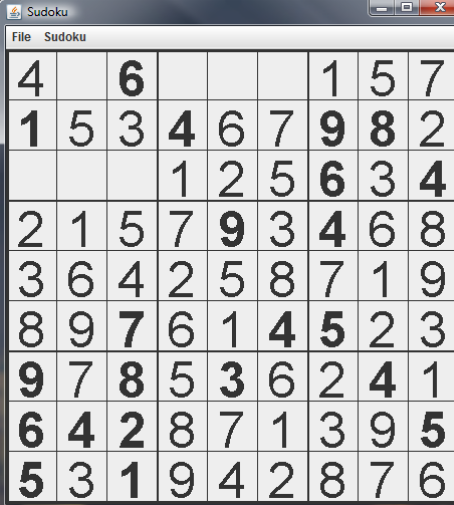
4		6				1	5	7
1	5	3	4	6	7	9	8	2
			1	2	5	6	3	4
2	1	5	7	9	3	4	6	8
3	6	4	2	5	8	7	1	9
8	9	7	6	1	4	5	2	3
9	7	8	5	3	6	2	4	1
6	4	2	8	7	1	3	9	5
5	3	1	9	4	2	8	7	6

combinator parsing

Sudoku: odczyt z pliku tekstowego

```
4 . [6] . . . 1 5 7
[1] 5 3 [4] 6 7 [9] [8] 2
. . . 1 2 5 [6] 3 [4]
2 1 5 7 [9] 3 [4] 6 8
3 6 4 2 5 8 7 1 9
8 9 [7] 6 1 [4] [5] 2 3
[9] 7 [8] 5 [3] 6 2 [4] 1
[6] [4] [2] 8 7 1 3 9 [5]
[5] 3 [1] 9 4 2 8 7 6
```

Parser



A screenshot of a window titled "Sudoku" with a menu bar containing "File" and "Sudoku". The window displays a 9x9 grid representing a solved Sudoku puzzle. The numbers in the grid correspond to the text above, with some numbers (6, 4, 9, 4, 5, 2, 4, 1, 5, 3, 1) rendered in a larger, bold font to highlight the results of the combinator parsing process.

4		6				1	5	7
1	5	3	4	6	7	9	8	2
			1	2	5	6	3	4
2	1	5	7	9	3	4	6	8
3	6	4	2	5	8	7	1	9
8	9	7	6	1	4	5	2	3
9	7	8	5	3	6	2	4	1
6	4	2	8	7	1	3	9	5
5	3	1	9	4	2	8	7	6

combinator parsing

Sudoku: odczyt z pliku tekstowego

```
4 . [6] . . . 1 5 7
[1] 5 3 [4] 6 7 [9] [8] 2
. . . 1 2 5 [6] 3 [4]
2 1 5 7 [9] 3 [4] 6 8
3 6 4 2 5 8 7 1 9
8 9 [7] 6 1 [4] [5] 2 3
[9] 7 [8] 5 [3] 6 2 [4] 1
[6] [4] [2] 8 7 1 3 9 [5]
[5] 3 [1] 9 4 2 8 7 6
```

**Input
Reader [Char]**

Parser [Array [Field]]

Array [Field]

combinator parsing

Sudoku: odczyt z pliku tekstowego

```
4 . [6] . . . 1 5 7
[1] 5 3 [4] 6 7 [9] [8] 2
. . . 1 2 5 [6] 3 [4]
2 1 5 7 [9] 3 [4] 6 8
3 6 4 2 5 8 7 1 9
8 9 [7] 6 1 [4] [5] 2 3
[9] 7 [8] 5 [3] 6 2 [4] 1
[6] [4] [2] 8 7 1 3 9 [5]
[5] 3 [1] 9 4 2 8 7 6
```

**Input
Reader [Char]**

Parser [Array [Field]]

ParseResult [Array [Field]]

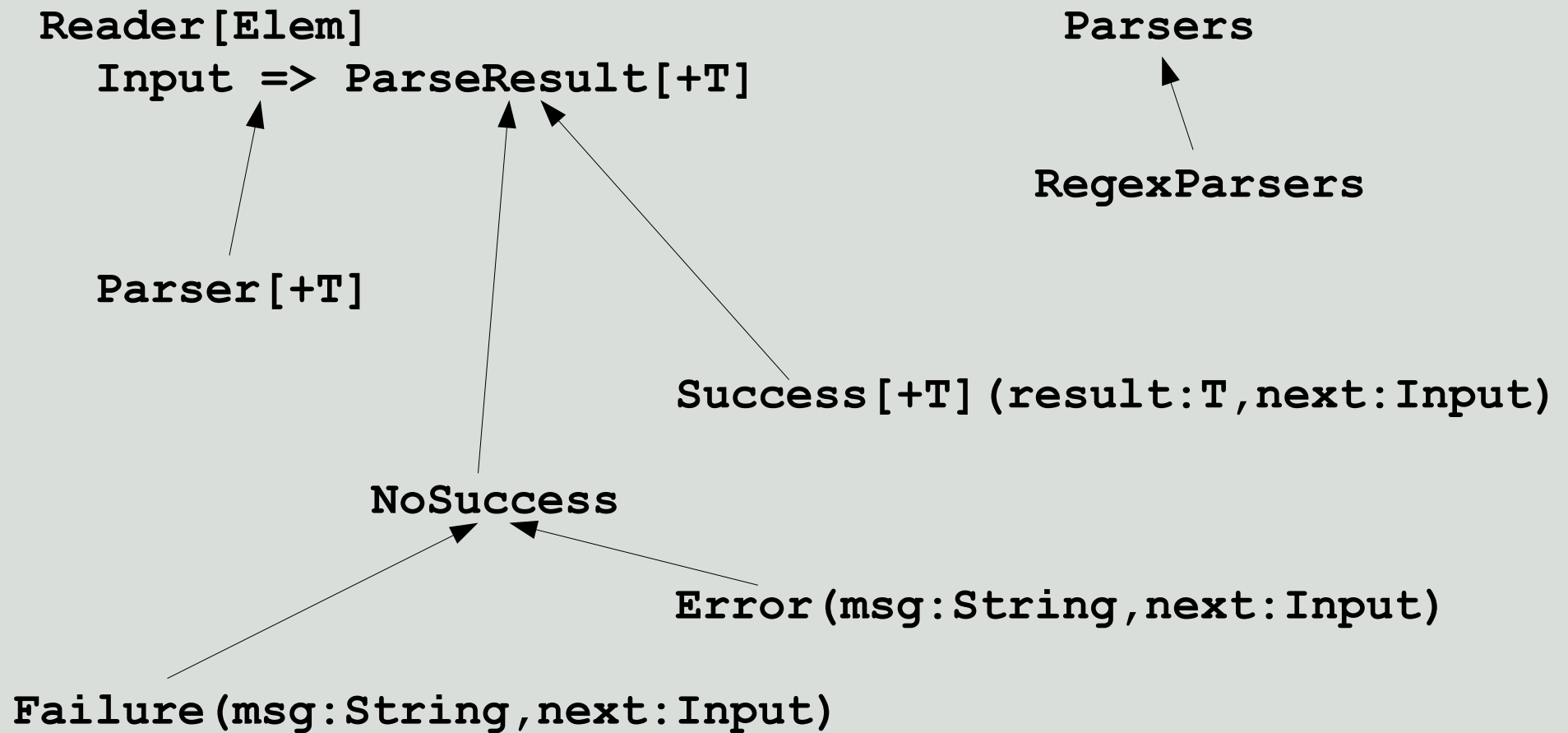
Failure, Error

Success

String

Array [Field]

combinator parsing typy



combinator parsing

Sudoku: krok 1 - gramatyka

`<board> ::= <field> <field> ... <field> (81 powtórzeń)`

`<field> ::= <empty> | <fixed> | <variable>`

`<empty> ::= .`

`<fixed> ::= [<value>]`

`<variable> ::= <value>`

`<value> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9`

combinator parsing

Sudoku: krok 2 - od gramatyki do scali

```
<board> ::= <field> <field> ... <field> (81 powtórzeń)
  def board: Parser[Any] = repN(81, field)
<field> ::= <empty> | <fixed> | <variable>
  def field: Parser[Any] = empty | fixed | variable
<empty> ::= .
  def empty: Parser[Any] = "."
<fixed> ::= [ <value> ]
  def fixed: Parser[Any] = "[" ~ value ~ "]"
<variable> ::= <value>
  def variable: Parser[Any] = value
<value> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
  def value: Parser[Any] = "1" | "2" | "3" | "4" | "5" |
    "6" | "7" | "8" | "9"
```

combinator parsing

Sudoku: krok 3 - pierwsza wersja parsera

```
object SudokuParsers1 extends RegexParsers {
  def apply(content: String) = parseAll(board, content)
  def board: Parser[Any] = repN(81, field)

  def field: Parser[Any] = empty | fixed | variable
  def empty: Parser[Any] = "."
  def fixed: Parser[Any] = "[" ~ value ~ "]"

  def variable: Parser[Any] = value

  def value: Parser[Any] = "1" | "2" | "3" | "4" | "5" |
    "6" | "7" | "8" | "9"
}
```

combinator parsing

Sudoku: krok 3 - testy

```
scala> import sudoku._  
import sudoku._
```

```
scala> import SudokuParsers1._  
import SudokuParsers1._
```

```
scala> import scala.util.parsing.input._  
import scala.util.parsing.input._
```

```
scala> implicit def s2r(s: String)=new CharSequenceReader(s)  
s2r: (s: String)scala.util.parsing.input.CharSequenceReader
```

combinator parsing

Sudoku: krok 3 - testy

```
scala> val parser1 = SudokuParsers1.fixed
parser1: sudoku.SudokuParsers1.Parser[Any] = Parser (~)
```

```
scala> var reader1 = new CharSequenceReader("[4]")
reader1: scala.util.parsing.input.CharSequenceReader = scala.
util.parsing.input.CharSequenceReader@11a6631
```

```
scala> val parseResult1 = parser1(reader1)
parseResult1: sudoku.SudokuParsers1.ParseResult[Any] = [1.4]
parsed: (([~4)~])
```

```
scala> val Success(result1, _) = parseResult1
result1: Any = (([~4)~])
```

```
scala> val Success(result1, _) = fixed("[4]")
result1: Any = (([~4)~])
```

combinator parsing

Sudoku: krok 3 - testy

```
scala> val r = fixed("[4)")
```

```
r: sudoku.SudokuParsers1.ParseResult[Any] =  
[1.3] failure: `]' expected but `)' found
```

```
[4)
```

```
^
```

```
scala> val r = field(".")
```

```
r: sudoku.SudokuParsers1.ParseResult[Any] = [1.2] parsed: .
```

```
scala> val r = field("2")
```

```
r: sudoku.SudokuParsers1.ParseResult[Any] = [1.2] parsed: 2
```

```
scala> val r = field("[5]")
```

```
r: sudoku.SudokuParsers1.ParseResult[Any] = [1.4] parsed: (([  
~5)~])
```

combinator parsing

Sudoku: krok 3 - testy

```
scala> val input: String = "" 4 . [6] . . . 1 5 7
| [1] 5 3 [4] 6 7 [9][8] 2
| . . . 1 2 5 [6] 3 [4]
| 2 1 5 7 [9] 3 [4] 6 8
| 3 6 4 2 5 8 7 1 9
| 8 9 [7] 6 1 [4][5] 2 3
| [9] 7 [8] 5 [3] 6 2 [4] 1
| [6][4][2] 8 7 1 3 9 [5]
| [5] 3 [1] 9 4 2 8 7 6 ""
```

```
input: String =
4 . [6] . . . 1 5 7
[1] 5 3 [4] 6 7 [9][8] 2
. . . 1 2 5 [6] 3 [4]
2 1 5 7 [9] 3 [4] 6 8
3 6 4 2 5 8 7 1 9
8 9 [7] 6 1 [4][5] 2 3
[9] 7 [8] 5 [3] 6 2 [4] 1
[6][4][2] 8 7 1 3 9 [5]
[5] 3 [1] 9 4 2 8 7 6
```

combinator parsing

Sudoku: krok 3 - testy

```
scala> val result = SudokuParsers1(input)
result: sudoku.SudokuParsers1.ParseResult[Any] = [9.28] parse
d: List(4, ., (([~6]~)), ., ., ., 1, 5, 7, (([~1]~)), 5, 3, (
([~4]~)), 6, 7, (([~9]~)), (([~8]~)), 2, ., ., ., 1, 2, 5, ((
[~6]~)), 3, (([~4]~)), 2, 1, 5, 7, (([~9]~)), 3, (([~4]~)), 6
, 8, 3, 6, 4, 2, 5, 8, 7, 1, 9, 8, 9, (([~7]~)), 6, 1, (([~4]
~)), (([~5]~)), 2, 3, (([~9]~)), 7, (([~8]~)), 5, (([~3]~)),
6, 2, (([~4]~)), 1, (([~6]~)), (([~4]~)), (([~2]~)), 8, 7, 1,
3, 9, (([~5]~)), (([~5]~)), 3, (([~1]~)), 9, 4, 2, 8, 7, 6)
```

```
scala> val result = SudokuParsers1("[1] . 4 (9) [3]")
result: sudoku.SudokuParsers1.ParseResult[Any] =
[1.9] failure: `9' expected but ` ' found
```

```
[1] . 4 (9) [3]
      ^
```

combinator parsing

Sudoku: krok 4 - druga wersja parsera

```
object SudokuParsers2 extends RegexParsers {
  def apply(content: String) = parseAll(board, content)
  def board: Parser[Array[Field]] = repN(81, field) ^^
    { _.toArray }
  def field: Parser[Field] = empty | fixed | variable
  def empty: Parser[Field] = "." ^^ EmptyField
  def fixed: Parser[Field] = "[" ~> value <~ "]" ^^
    { FixedField(_) }
  def variable: Parser[Field] = value ^^
    { VariableField(_) }
  def value: Parser[Int] = ("1"|"2"|"3"|"4"|"5"|"
    "6"|"7"|"8"|"9") ^^ (_.toInt)
}
```

combinator parsing

Sudoku: krok 4 - testy

```
scala> import sudoku._  
import sudoku._
```

```
scala> import SudokuParsers2._  
import SudokuParsers2._
```

```
scala> import scala.util.parsing.input._  
import scala.util.parsing.input._
```

```
scala> implicit def s2r(s: String)=new CharSequenceReader(s)  
s2r: (s: String)scala.util.parsing.input.CharSequenceReader
```

combinator parsing

Sudoku: krok 4 - testy

```
scala> val r = field(".")
```

```
r: sudoku.SudokuParsers2.ParseResult[sudoku.Field] = [1.2] pa  
rsed: EmptyField
```

```
scala> val r = field("2")
```

```
r: sudoku.SudokuParsers2.ParseResult[sudoku.Field] = [1.2] pa  
rsed: VariableField(2)
```

```
scala> val r = field("[5]")
```

```
r: sudoku.SudokuParsers2.ParseResult[sudoku.Field] = [1.4] pa  
rsed: FixedField(5)
```

combinator parsing

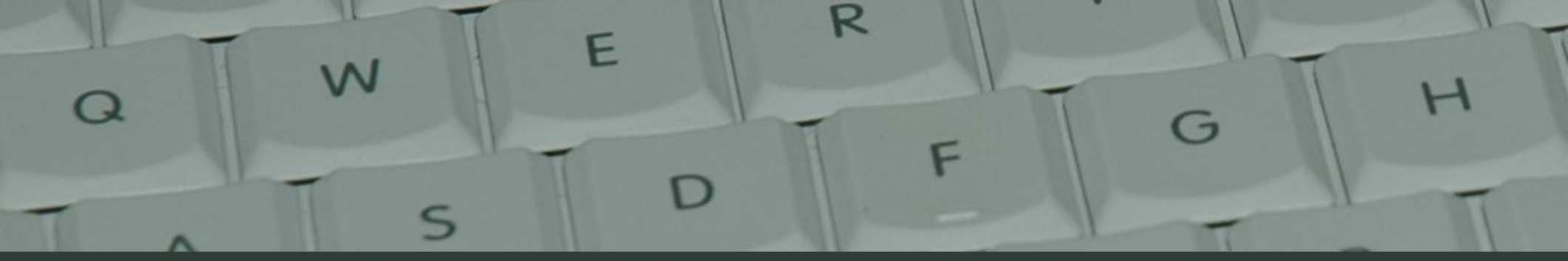
Sudoku: krok 4 - testy

```
scala> val Success(result, _) = SudokuParsers2(input)
result: Array[sudoku.Field] = Array(VariableField(4), EmptyField, FixedField(6), EmptyField, EmptyField, EmptyField, VariableField(1), VariableField(5), VariableField(7), FixedField(1), VariableField(5), VariableField(3), FixedField(4), VariableField(6), VariableField(7), FixedField(9), FixedField(8), VariableField(2), EmptyField, EmptyField, EmptyField, VariableField(1), VariableField(2), VariableField(5), FixedField(6), VariableField(3), FixedField(4), VariableField(2), VariableField(1), VariableField(5), VariableField(7), FixedField(9), VariableField(3), FixedField(4), VariableField(6), VariableField(8), VariableField(3), VariableField(6), VariableField(4), VariableField(2), VariableField(5), VariableField(8), VariableField(7), VariableField(1), VariableField(9), VariableField(8), Va...
```

combinator parsing

Sudoku: krok 5 - odczyt pliku i parsowanie

```
object TxtBoardReader extends BoardReader {  
  override def load(file: File, destArray: Array[Field]) {  
    import SudokuParsers2._  
    val content = scala.io.Source.fromFile(file).  
      addString(new StringBuilder).toString  
    SudokuParsers2(content) match {  
      case Success(newBoard, _) =>  
        Array.copy(newBoard, 0, destArray, 0, 81)  
      case x => MessageDialog(x.toString).open  
    }  
  }  
}
```



actors



actors

- Co to jest "aktor"?
- trait scala.actors.Actor
- Metoda act()
- Wysyłanie i odbieranie komunikatów
 - wysyłanie: m.in. !, !?, forward, reply
 - odbieranie: m.in. receive, react, !?

actors

przykład: GuessClient1

```
class GuessClient1(server: Actor) extends Actor {
  def act() {
    println("GuessClient1: starting")
    val solution = guess(5)
    println("GuessClient1: solution found: "+solution)
  }
  private def guess(n: Int): Int = {
    println("GuessClient1: trying: " + n)
    server ! n
    val correction = receive {
      case "correct" => 0
      case "too big" => -1
      case "too small" => 1
    }
    if (correction == 0) n else guess(n + correction)
  }
}
```

actors

przykład: GuessServer1

```
class GuessServer1 extends Actor {
  def act() {
    println("GuessServer1: starting")
    respond(scala.util.Random.nextInt(10))
  }
  private def respond(n: Int) {
    val (client, response) = receive {
      case x:Int if x > n => (sender, "too big")
      case x:Int if x < n => (sender, "too small")
      case x:Int => (sender, "correct")
    }
    println("GuessServer1: "+response)
    client ! response
    if (response != "correct") respond(n)
  }
}
```

actors

GuessServer1 i GuessClient1

```
scala> val server = new GuessServer1
server: GuessServer1 = GuessServer1@104f8eb

scala> val client = new GuessClient1(server)
client: GuessClient1 = GuessClient1@c84051

scala> server.start(); client.start()
GuessServer1: starting

scala> GuessClient1: starting
GuessClient1: trying: 5
GuessServer1: too big
GuessClient1: trying: 4
GuessServer1: correct
GuessClient1: solution found: 4
```

actors

przykład: GuessClient2

```
class GuessClient2(server: Actor) extends Actor {
  def act() {
    println("GuessClient2: starting")
    val solution = guess(5)
    println("GuessClient2: solution found: "+solution)
  }
  private def guess(n: Int): Int = {
    println("GuessClient2: trying: " + n)
    val response = server !? n
    println("GuessClient2: got response: " + response)
    val correction = response match {
      case "correct" => 0
      case "too big" => -1
      case "too small" => 1 }
    if (correction == 0) n else guess(n + correction)
  }
}
```

actors

przykład: GuessServer2

```
class GuessServer2 extends Actor {
  val n = scala.util.Random.nextInt(10)
  def act() {
    println("GuessServer2: starting")
    awaitGuess
    println("GuessServer2: ending")
  }
  private def awaitGuess {
    react {
      case x:Int if x > n => reply("too big"); awaitGuess
      case x:Int if x < n => reply("too small"); awaitGuess
      case x:Int => reply("correct")
    }
  }
}
```

actors

GuessServer2 i GuessClient2

```
scala> val server = new GuessServer2
server: GuessServer2 = GuessServer2@1c3d34b

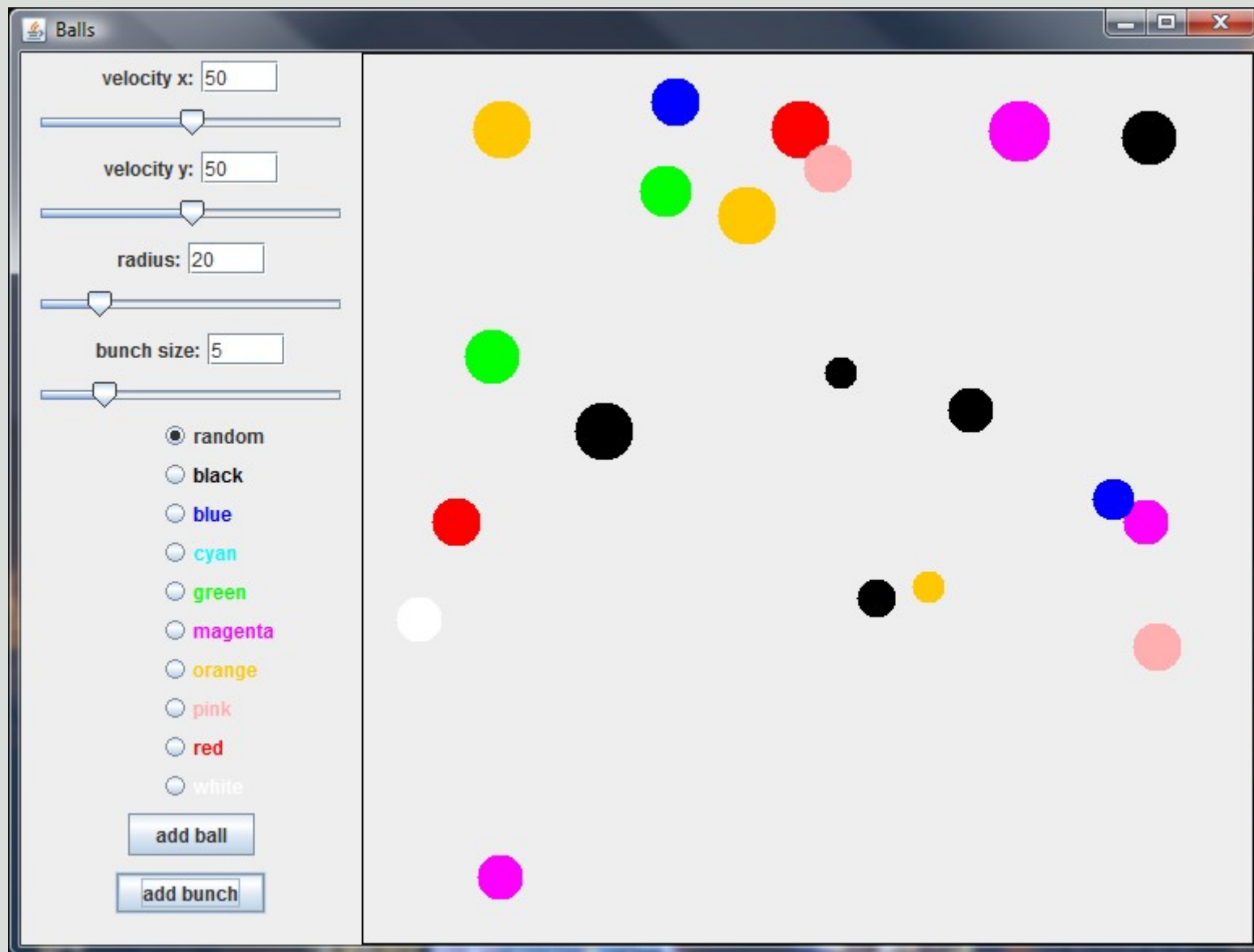
scala> val client = new GuessClient2(server)
client: GuessClient2 = GuessClient2@1ba3c1f

scala> server.start(); client.start()
GuessServer2: starting

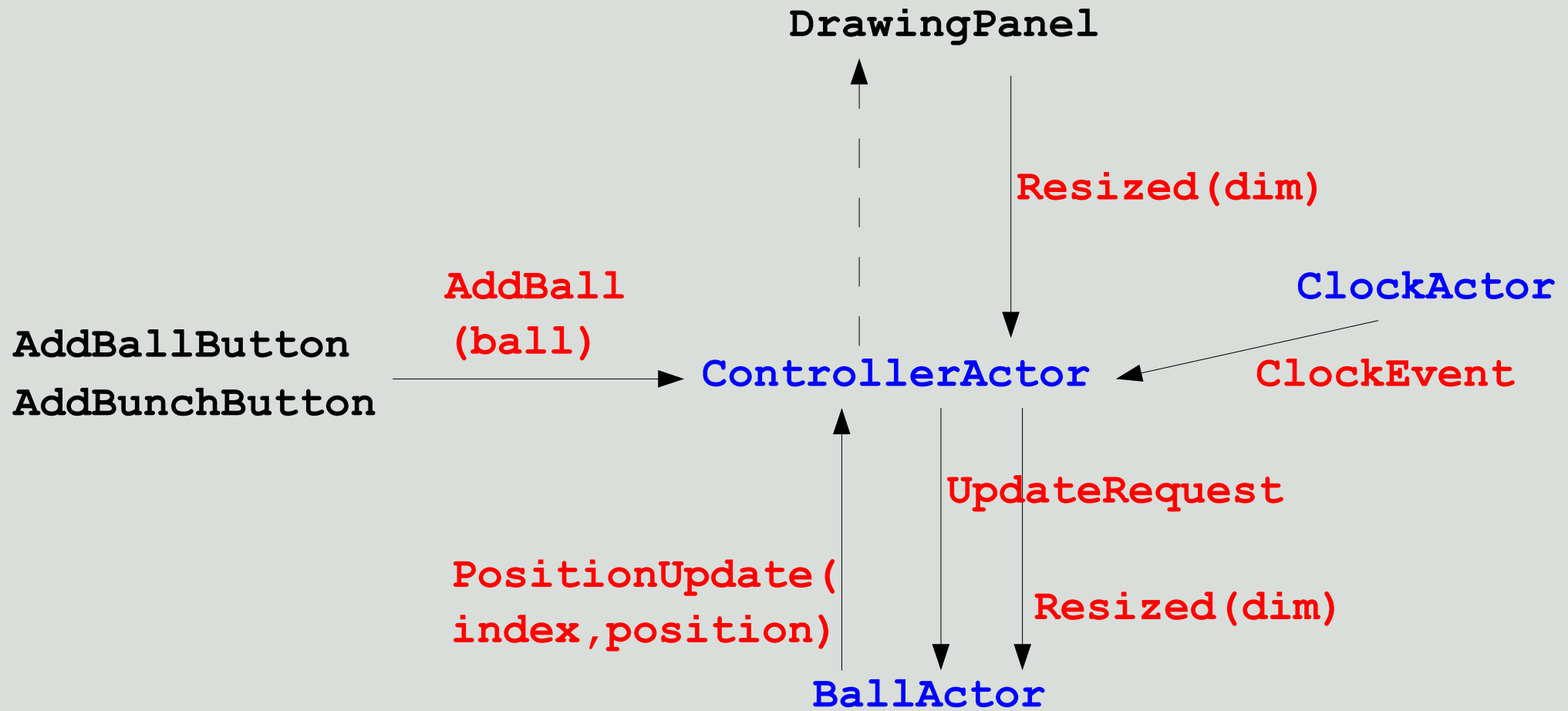
scala> GuessClient2: starting
GuessClient2: trying: 5
GuessClient2: got response: too small
GuessClient2: trying: 6
GuessClient2: got response: correct
GuessClient2: solution found: 6
```

actors

przykład: Balls



actors Balls



actors

ClockActor

```
class ClockActor(sleepTime: Int, controller: Actor)
  extends Actor {
    override def act() {
      while(true) {
        controller ! ClockEvent
        Thread.sleep(sleepTime)
      }
    }
  }
}
```

actors

BallActor

```
class BallActor(ball: Ball,  
  position: Point, index: Int,  
  private[this] var dim: Dimension)  
extends Actor {  
  private[this] var xPos = position.x.toDouble  
  private[this] var yPos = position.y.toDouble  
  (...)   
  def updatePosition {  
  (...)   
    val dx = (ball.vx * intervalTime / 1000.0).abs  
    val dy = (ball.vy * intervalTime / 1000.0).abs  
    xPos += dx * signx  
    yPos += dy * signy  
  (...)   
  }
```

actors

BallActor

```
val messageHandler: PartialFunction[Any,Unit] = {  
  
  case UpdateRequest =>  
    updatePosition  
    reply(PositionUpdate(index,  
      new Point(xPos.toInt, yPos.toInt)))  
  
  case Resized(dim) => this.dim = dim  
  
}  
  
override def act() = loop { react(messageHandler) }  
}
```

actors

ControllerActor

```
class ControllerActor(drawingPanel: DrawingPanel,  
  clockInterval: Int,  
  (...)  
extends Actor {  
  (...)  
  private[this] val positions = mutable.Map[Int, Point]()  
  private[this] val actors = mutable.Map[Int, Actor]()  
  private[this] val handlers =  
    ListBuffer[PartialFunction[Any, Unit]]()  
  
  override def act() {  
    (...)  
    new ClockActor(clockInterval, this).start()  
    while(true) { receive(handlers reduceLeft (_ orElse _)) }  
  }  
}
```

actors

ControllerActor

```
handlers += {
  case AddBall(ball) =>
  (...)
    actors(ballsCounter) =
      createActor(ball, position, ballsCounter, dim).start()
  (...)
}
def createActor(ball: Ball, position: Point,
  index: Int, dim: Dimension) =
  new BallActor(ball, position, index, dim)
  (...)
```

actors

ControllerActor

```
handlers += {
  case ClockEvent =>
    updateDrawingPanelPositions
    for (j <- 0 to ballsCounter-1)
      actors(j) ! UpdateRequest
}
def updateDrawingPanelPositions {
  val pos2 = immutable.Map[Int, Point]() ++ positions
  Swing.onEDT { drawingPanel.updatePositions(pos2) }
}
```

actors

ControllerActor

```
handlers += {  
  case PositionUpdate(index, pos) =>  
    positions(index) = pos  
}
```

```
handlers += {  
  case msg @ Resized(dim) =>  
    this.dim = dim  
    for (j <- 0 to ballsCounter-1)  
      actors(j) forward msg  
}
```

```
}
```

actors wątki

- liczba aktorów = 2 + liczba kulek
- balls.App

```
override def act() = loop { react(messageHandler) }
```

- balls.App2

```
override def act() =  
  while(true) { receive(messageHandler) }
```

actors

przykład: Client, Server

Balls client

Server: localhost
Port: 9000
Actor: 'BallsControllerActor'

velocity x: 50

velocity y: 50

radius: 20

bunch size: 5

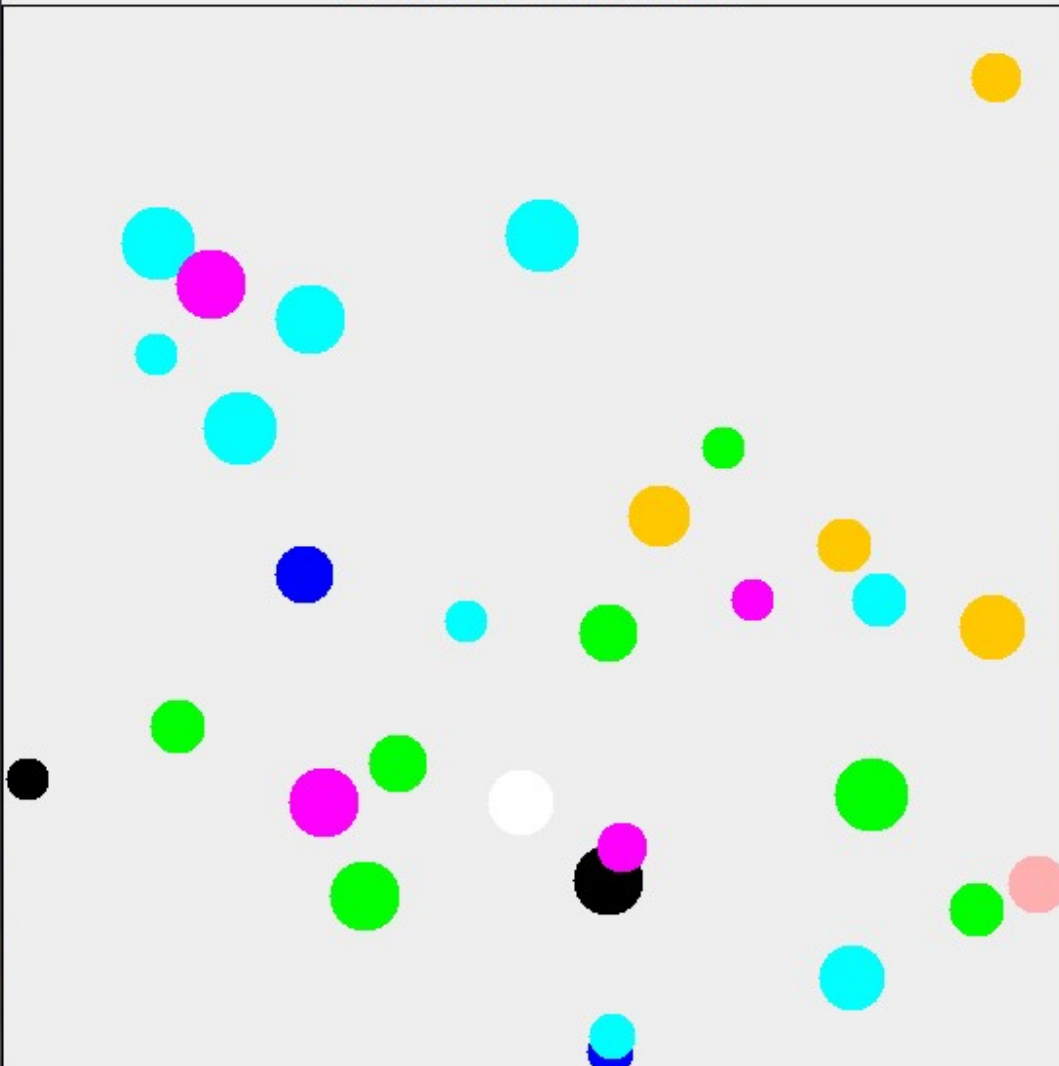
random
 black
 blue
 cyan
 green
 magenta
 orange
 pink
 red
 white

add ball

add bunch

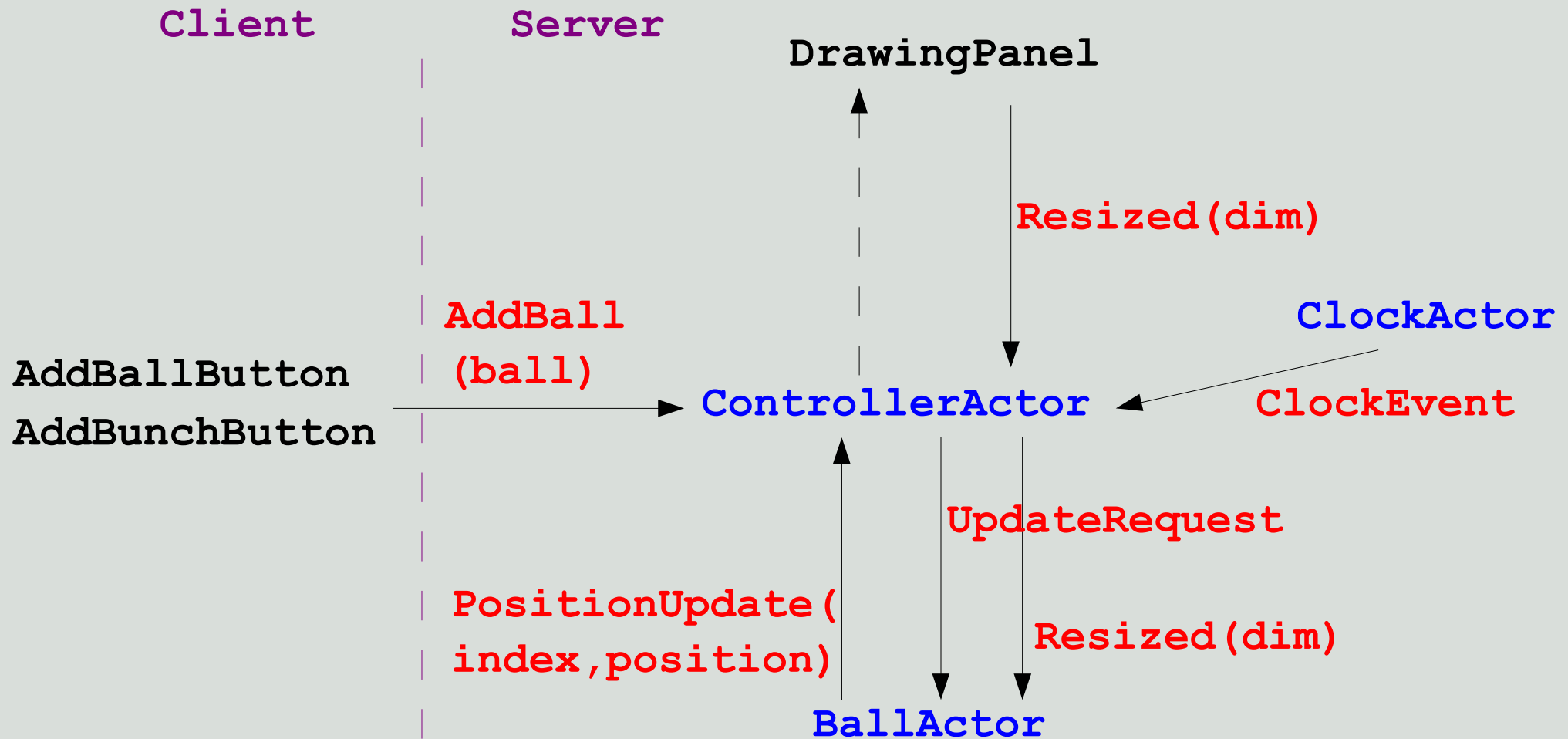
Balls server

Port: 9000 Actor: 'BallsControllerActor'



actors

Client, Server



actors Client

```
object Client extends AbstractBallsApp("Balls client") {  
  override def createController =  
    RemoteActor.select(Node(serverParam, portParam),  
      actorParam)  
  (...)  
}
```

```
class ControlPanel(controller: => AbstractActor,  
  (...)  
  case ButtonClicked(AddBallButton) =>  
    controller ! AddBall(new Ball(  
      ColorComponent.color,  
      RadiusField.text.toDouble,  
      VelocityXField.text.toDouble,  
      VelocityYField.text.toDouble))
```

actors

ControllerActor

```
class ControllerActor(val drawingPanel: DrawingPanel,  
  clockInterval: Int,  
  remoteActorParams: Option[(Int, Symbol)] = None)  
extends Actor {  
  (...)  
  override def act() {  
    for ((port, symbol) <- remoteActorParams) {  
      RemoteActor.alive(port)  
      RemoteActor.register(symbol, Actor.self)  
    }  
  }  
  (...)  
}
```

pytania?

