計算機科学第二

第三回 Readable Code (2)

本日の内容

- Item 56: Adhere to generally accepted naming conventions
- Item 45: Minimize the scope of local variables
- Item 49: Prefer primitive types to boxed primitives

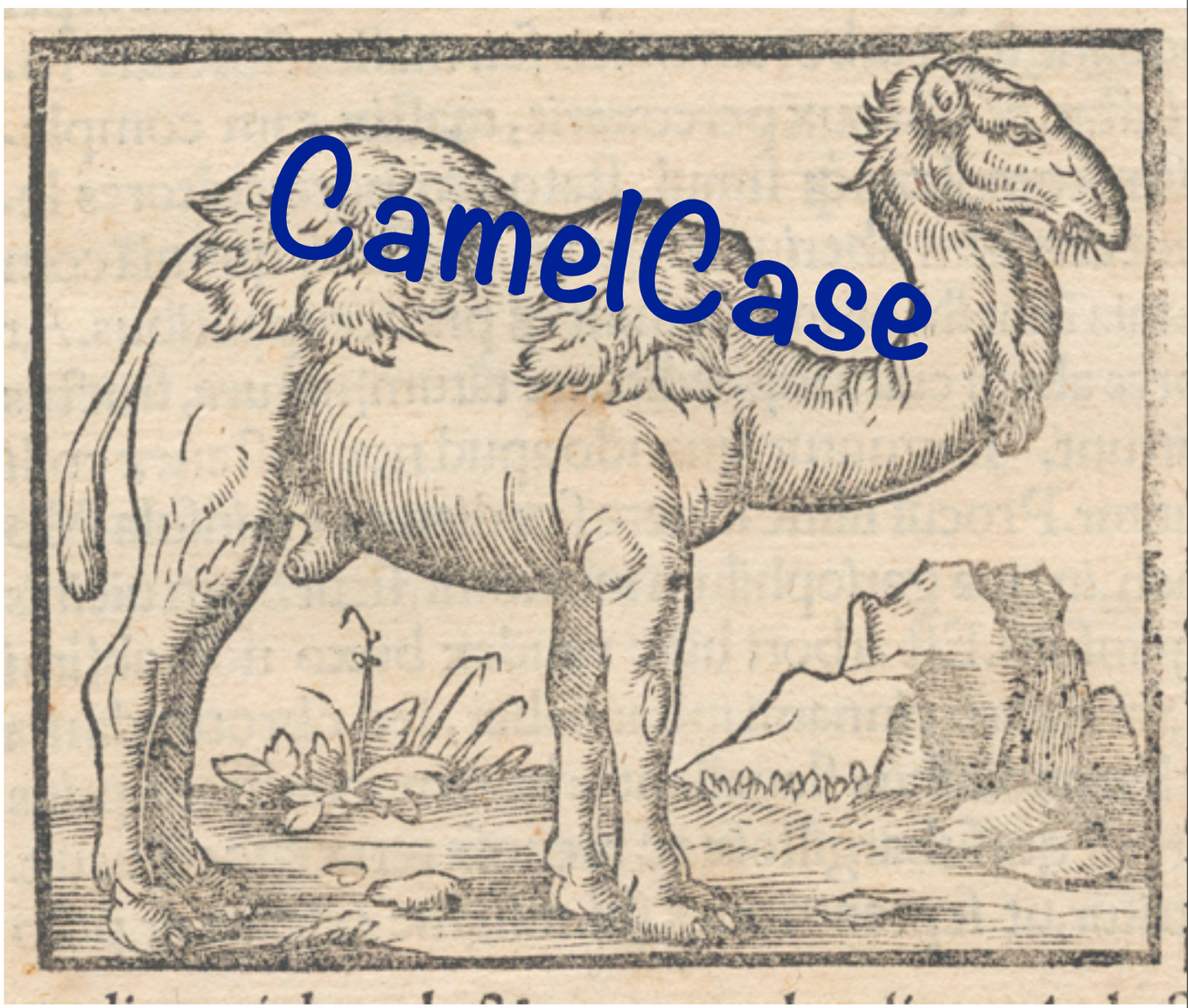
Adhere to generally accepted naming conventions

パッケージ名

- ドメイン名の逆順.パッケージ名
  - **org.junit.runner**
  - **jp.ac.titech.is.cs1.lecture03**
- java, javax で始まるパッケージは例外
  - **java.lang, java.util, java.util.regex, javax.swing, javax.swing.plaf.metal**

クラスとインタフェイス

- 大文字で始める
  - String, Timer, Vector
  - CamelCase
  - FutureTask



メソッド，フィールド

- 小文字で始まる CamelCase
  - println, parseInt
- ただし，定数は_で区切られた大文字
  - WIDTH, HEIGHT
  - POSITIVE_INTEGER
  - final int WINDOW_WIDTH = 1024;

局所変数

- メソッドやフィールド名と同様に小文字で始まる CamlCase

- i, xref, houseNumber

型変数

- 一文字の大文字
  - T, U, V
- 一文字の大文字 + 数値
  - T1, T2, T3, ...

識別子の種類	例
パッケージ	org.junit.runner jp.ac.titech.is.wakita.cs1
クラス／インタフェイス	String, Time, LinkedList, GregorianCalendar
メソッド／フィールド	println, parseInt x, y, size, winHeight
定数	WIDTH, HEIGHT POSITIVE_INTEGER
局所変数	x, y
型変数	T, U, V, T1, T2, ...

名前のつけ方の傾向

メソッドの種類	命名法
booleanメソッド	is..., isDigit, isLeap, isPrime
返り値のあるメソッド	名詞で終わる car.speed(), person.age()
getter/setter	get.../set...

Minimize the scope of local variables

変数のスコープ

- 変数の有効範囲
  - コードのなかで変数を参照・代入できる箇所の範囲。

Javaの変数

- フィールド変数
- メソッドの引数
- 局所変数

フィールドのスコープ

```
36      protected void initialize(CSVFieldType types[], String sep) {
37          initialize(types);
38          this.sep = sep;
39      }
60              if (t == CSVFieldType.Integer) { set(i, Integer.parseInt(token)); continue; }
61              if (t == CSVFieldType.Double) {
62                  double d = token.length() == 0 ? 0. : Double.parseDouble(token);
63                  set(i, d);
64                  continue;
65              }
66              if (t == CSVFieldType.String) { set(i, token); continue; }
67          }
68          add();
69      }
70
71      private String sep = ",\\s*";
72
73      protected void parse(java.io.Reader r) throws java.io.IOException {
74          StringBuffer pat = new StringBuffer("\\s*");
75          for (int i = 0; i < types.length - 1; i++) pat.append(types[i].pattern() + sep);
76          pat.append(types[types.length - 1].pattern());
77          BufferedReader reader = new BufferedReader(r);
78          String line;
79          for (int lineNo = 0; (line = reader.readLine()) != null; lineNo++) {
80              parse(line, Pattern.compile(pat.toString()));
```

フィールドのスコープ

```
36      protected void initialize(CSVFieldType types[], String sep) {
37          initialize(types);
38          this.sep = sep;
39      }
60              if (t == CSVFieldType.Integer) { set(i, Integer.parseInt(token)); continue; }
61              if (t == CSVFieldType.Double) {
62                  double d = token.length() == 0 ? 0. : Double.parseDouble(token);
63                  set(i, d);
64                  continue;
65              }
66              if (t == CSVFieldType.String) { set(i, token); continue; }
67          }
68          add();
69      }
70
71      private String sep = ",\\s*";
72
73      protected void parse(java.io.Reader r) throws java.io.IOException {
74          StringBuffer pat = new StringBuffer("\\s*");
75          for (int i = 0; i < types.length - 1; i++) pat.append(types[i].pattern() + sep);
76          pat.append(types[types.length - 1].pattern());
77          BufferedReader reader = new BufferedReader(r);
78          String line;
79          for (int lineNo = 0; (line = reader.readLine()) != null; lineNo++) {
80              parse(line, Pattern.compile(pat.toString()));
```

メソッドのスコープ

- メソッドの本体全体

メソッドの引数のスコープ

```
51      protected void parse(String line, Pattern p) {
52          Matcher match = p.matcher(line);
53
54          if (!match.matches()) { System.err.println("Unmatched line: " + line); return; }
55
56          for (int i = 0; i < types.length; i++) {
57              CSVFieldType t = types[i];
58              String token = match.group(i + 1);
59              if (t == CSVFieldType.Ignore) continue;
60              if (t == CSVFieldType.Integer) { set(i, Integer.parseInt(token)); continue; }
61              if (t == CSVFieldType.Double) {
62                  double d = token.length() == 0 ? 0. : Double.parseDouble(token);
63                  set(i, d);
64                  continue;
65              }
66              if (t == CSVFieldType.String) { set(i, token); continue; }
67          }
68          add();
69      }
```

メソッドの引数のスコープ

```
51      protected void parse(String line, Pattern p) {
52          Matcher match = p.matcher(line);
53
54          if (!match.matches()) { System.err.println("Unmatched line: " + line); return; }
55
56          for (int i = 0; i < types.length; i++) {
57              CSVFieldType t = types[i];
58              String token = match.group(i + 1);
59              if (t == CSVFieldType.Ignore) continue;
60              if (t == CSVFieldType.Integer) { set(i, Integer.parseInt(token)); continue; }
61              if (t == CSVFieldType.Double) {
62                  double d = token.length() == 0 ? 0. : Double.parseDouble(token);
63                  set(i, d);
64                  continue;
65              }
66              if (t == CSVFieldType.String) { set(i, token); continue; }
67          }
68          add();
69      }
```

メソッドの引数のスコープ

```
51      protected void parse(String line, Pattern p) {
52          Matcher match = p.matcher(line);
53
54          if (!match.matches()) { System.err.println("Unmatched line: " + line); return; }
55
56          for (int i = 0; i < types.length; i++) {
57              CSVFieldType t = types[i];
58              String token = match.group(i + 1);
59              if (t == CSVFieldType.Ignore) continue;
60              if (t == CSVFieldType.Integer) { set(i, Integer.parseInt(token)); continue; }
61              if (t == CSVFieldType.Double) {
62                  double d = token.length() == 0 ? 0. : Double.parseDouble(token);
63                  set(i, d);
64                  continue;
65              }
66              if (t == CSVFieldType.String) { set(i, token); continue; }
67          }
68          add();
69      }
```

メソッドの引数のスコープ

```
51      protected void parse(String line, Pattern p) {
52          Matcher match = p.matcher(line);
53
54          if (!match.matches()) { System.err.println("Unmatched line: " + line); return; }
55
56          for (int i = 0; i < types.length; i++) {
57              CSVFieldType t = types[i];
58              String token = match.group(i + 1);
59              if (t == CSVFieldType.Ignore) continue;
60              if (t == CSVFieldType.Integer) { set(i, Integer.parseInt(token)); continue; }
61              if (t == CSVFieldType.Double) {
62                  double d = token.length() == 0 ? 0. : Double.parseDouble(token);
63                  set(i, d);
64                  continue;
65              }
66              if (t == CSVFieldType.String) { set(i, token); continue; }
67          }
68          add();
69      }
```

局所変数のスコープ

- 変数宣言を囲む最内ブロックのうち、変数宣言に続く場所
- ※ Javaのブロック: 対応する { と } で囲われた範囲

局所変数のスコープ

```
51      protected void parse(String line, Pattern p) {
52          Matcher match = p.matcher(line);
53
54          if (!match.matches()) { System.err.println("Unmatched line: " + line); return; }
55
56          for (int i = 0; i < types.length; i++) {
57              CSVFieldType t = types[i];
58              String token = match.group(i + 1);
59              if (t == CSVFieldType.Ignore) continue;
60              if (t == CSVFieldType.Integer) { set(i, Integer.parseInt(token)); continue; }
61              if (t == CSVFieldType.Double) {
62                  double d = token.length() == 0 ? 0. : Double.parseDouble(token);
63                  set(i, d);
64                  continue;
65              }
66              if (t == CSVFieldType.String) { set(i, token); continue; }
67          }
68          add();
69      }
```

局所変数のスコープ

```
51      protected void parse(String line, Pattern p) {
52          Matcher match = p.matcher(line);
53
54          if (!match.matches()) { System.err.println("Unmatched line: " + line); return; }
55
56          for (int i = 0; i < types.length; i++) {
57              CSVFieldType t = types[i];
58              String token = match.group(i + 1);
59              if (t == CSVFieldType.Ignore) continue;
60              if (t == CSVFieldType.Integer) { set(i, Integer.parseInt(token)); continue; }
61              if (t == CSVFieldType.Double) {
62                  double d = token.length() == 0 ? 0. : Double.parseDouble(token);
63                  set(i, d);
64                  continue;
65              }
66              if (t == CSVFieldType.String) { set(i, token); continue; }
67          }
68          add();
69      }
```

局所変数のスコープ

```
51      protected void parse(String line, Pattern p) {
52          Matcher match = p.matcher(line);
53
54          if (!match.matches()) { System.err.println("Unmatched line: " + line); return; }
55
56          for (int i = 0; i < types.length; i++) {
57              CSVFieldType t = types[i];
58              String token = match.group(i + 1);
59              if (t == CSVFieldType.Ignore) continue;
60              if (t == CSVFieldType.Integer) { set(i, Integer.parseInt(token)); continue; }
61              if (t == CSVFieldType.Double) {
62                  double d = token.length() == 0 ? 0. : Double.parseDouble(token);
63                  set(i, d);
64                  continue;
65              }
66              if (t == CSVFieldType.String) { set(i, token); continue; }
67          }
68          add();
69      }
```

局所変数のスコープ

```
51      protected void parse(String line, Pattern p) {
52          Matcher match = p.matcher(line);
53
54          if (!match.matches()) { System.err.println("Unmatched line: " + line); return; }
55
56          for (int i = 0; i < types.length; i++) {
57              CSVFieldType t = types[i];
58              String token = match.group(i + 1);
59              if (t == CSVFieldType.Ignore) continue;
60              if (t == CSVFieldType.Integer) { set(i, Integer.parseInt(token)); continue; }
61              if (t == CSVFieldType.Double) {
62                  double d = token.length() == 0 ? 0. : Double.parseDouble(token);
63                  set(i, d);
64                  continue;
65              }
66              if (t == CSVFieldType.String) { set(i, token); continue; }
67          }
68          add();
69      }
```

局所変数のスコープ

```
51      protected void parse(String line, Pattern p) {
52          Matcher match = p.matcher(line);
53
54          if (!match.matches()) { System.err.println("Unmatched line: " + line); return; }
55
56          for (int i = 0; i < types.length; i++) {
57              CSVFieldType t = types[i];
58              String token = match.group(i + 1);
59              if (t == CSVFieldType.Ignore) continue;
60              if (t == CSVFieldType.Integer) { set(i, Integer.parseInt(token)); continue; }
61              if (t == CSVFieldType.Double) {
62                  double d = token.length() == 0 ? 0. : Double.parseDouble(token);
63                  set(i, d);
64                  continue;
65              }
66              if (t == CSVFieldType.String) { set(i, token); continue; }
67          }
68          add();
69      }
```

局所変数のスコープ

```
51      protected void parse(String line, Pattern p) {
52          Matcher match = p.matcher(line);
53
54          if (!match.matches()) { System.err.println("Unmatched line: " + line); return; }
55
56          for (int i = 0; i < types.length; i++) {
57              CSVFieldType t = types[i];
58              String token = match.group(i + 1);
59              if (t == CSVFieldType.Ignore) continue;
60              if (t == CSVFieldType.Integer) { set(i, Integer.parseInt(token)); continue; }
61              if (t == CSVFieldType.Double) {
62                  double d = token.length() == 0 ? 0. : Double.parseDouble(token);
63                  set(i, d);
64                  continue;
65              }
66              if (t == CSVFieldType.String) { set(i, token); continue; }
67          }
68          add();
69      }
```

局所変数のスコープ

```
51      protected void parse(String line, Pattern p) {
52          Matcher match = p.matcher(line);
53
54          if (!match.matches()) { System.err.println("Unmatched line: " + line); return; }
55
56          for (int i = 0; i < types.length; i++) {
57              CSVFieldType t = types[i];
58              String token = match.group(i + 1);
59              if (t == CSVFieldType.Ignore) continue;
60              if (t == CSVFieldType.Integer) { set(i, Integer.parseInt(token)); continue; }
61              if (t == CSVFieldType.Double) {
62                  double d = token.length() == 0 ? 0. : Double.parseDouble(token);
63                  set(i, d);
64                  continue;
65              }
66              if (t == CSVFieldType.String) { set(i, token); continue; }
67          }
68          add();
69      }
```

局所変数のスコープ

```
51      protected void parse(String line, Pattern p) {
52          Matcher match = p.matcher(line);
53
54          if (!match.matches()) { System.err.println("Unmatched line: " + line); return; }
55
56          for (int i = 0; i < types.length; i++) {
57              CSVFieldType t = types[i];
58              String token = match.group(i + 1);
59              if (t == CSVFieldType.Ignore) continue;
60              if (t == CSVFieldType.Integer) { set(i, Integer.parseInt(token)); continue; }
61              if (t == CSVFieldType.Double) {
62                  double d = token.length() == 0 ? 0. : Double.parseDouble(token);
63                  set(i, d);
64                  continue;
65              }
66              if (t == CSVFieldType.String) { set(i, token); continue; }
67          }
68          add();
69      }
```

forループ変数のスコープ

- for文のTest部、 Increment部、 本体

ループ変数のスコープ

```
51      protected void parse(String line, Pattern p) {
52          Matcher match = p.matcher(line);
53
54          if (!match.matches()) { System.err.println("Unmatched line: " + line); return; }
55
56          for (int i = 0; i < types.length; i++) {
57              CSVFieldType t = types[i];
58              String token = match.group(i + 1);
59              if (t == CSVFieldType.Ignore) continue;
60              if (t == CSVFieldType.Integer) { set(i, Integer.parseInt(token)); continue; }
61              if (t == CSVFieldType.Double) {
62                  double d = token.length() == 0 ? 0. : Double.parseDouble(token);
63                  set(i, d);
64                  continue;
65              }
66              if (t == CSVFieldType.String) { set(i, token); continue; }
67          }
68          add();
69      }
```

ループ変数のスコープ

```
51      protected void parse(String line, Pattern p) {
52          Matcher match = p.matcher(line);
53
54          if (!match.matches()) { System.err.println("Unmatched line: " + line); return; }
55
56          for (int i = 0; i < types.length; i++) {
57              CSVFieldType t = types[i];
58              String token = match.group(i + 1);
59              if (t == CSVFieldType.Ignore) continue;
60              if (t == CSVFieldType.Integer) { set(i, Integer.parseInt(token)); continue; }
61              if (t == CSVFieldType.Double) {
62                  double d = token.length() == 0 ? 0. : Double.parseDouble(token);
63                  set(i, d);
64                  continue;
65              }
66              if (t == CSVFieldType.String) { set(i, token); continue; }
67          }
68          add();
69      }
```

スコープ最狭化の原則

- 変数は、それを使用する箇所の直前で宣言すること
- 宣言した変数は可能な限り初期化すること

スコープを限定する技術I

- 局所ブロックの利用
- for/while/if/switch がなくてもブロックは利用できる。

```
SDView(Container topPane) {
        JTabbedPane tabbedPane = new JTabbedPane();
        topPane.add(tabbedPane);

        {
                JPanel panel = new JPanel(false);
                panel.setLayout(new BorderLayout());

                JPanel toolbar = new JPanel();
                toolbar.setLayout(new FlowLayout(FlowLayout.LEFT));
                panel.add(toolbar, BorderLayout.NORTH);

                SDToolButtonFactory toolButtonFactory = new SDToolButtonFactory(model);
                toolbar.add(toolButtonFactory.createDrawTools());
                toolbar.add(toolButtonFactory.createSCTools());

                GPanel gpanel = new GPanel(model);
                model.setView(panel);
                SDEditorControl control = new SDEditorControl(model);
                gpanel.addMouseListener(control);
                gpanel.addMouseMotionListener(control);
                panel.add(gpanel);

                tabbedPane.addTab("Drawing", panel);
        }

        JPanel colorPanel = new JPanel(false);
        tabbedPane.addTab("Color", colorPanel);

        JPanel analyzePanel = new JPanel(false);
        tabbedPane.addTab("Analyze", analyzePanel);
}
```

```
SDView(Container topPane) {
        JTabbedPane tabbedPane = new JTabbedPane();
        topPane.add(tabbedPane);

        {
                JPanel panel = new JPanel(false);
                panel.setLayout(new BorderLayout());

                JPanel toolbar = new JPanel();
                toolbar.setLayout(new FlowLayout(FlowLayout.LEFT));
                panel.add(toolbar, BorderLayout.NORTH);

                SDToolButtonFactory toolButtonFactory = new SDToolButtonFactory(model);
                toolbar.add(toolButtonFactory.createDrawTools());
                toolbar.add(toolButtonFactory.createSCTools());

                GPanel gpanel = new GPanel(model);
                model.setView(panel);
                SDEditorControl control = new SDEditorControl(model);
                gpanel.addMouseListener(control);
                gpanel.addMouseMotionListener(control);
                panel.add(gpanel);

                tabbedPane.addTab("Drawing", panel);
        }

        JPanel colorPanel = new JPanel(false);
        tabbedPane.addTab("Color", colorPanel);

        JPanel analyzePanel = new JPanel(false);
        tabbedPane.addTab("Analyze", analyzePanel);
}
```

```
SDView(Container topPane) {
        JTabbedPane tabbedPane = new JTabbedPane();
        topPane.add(tabbedPane);

        {
            JPanel panel = new JPanel(false);
            panel.setLayout(new BorderLayout());

            JPanel toolbar = new JPanel();
            toolbar.setLayout(new FlowLayout(FlowLayout.LEFT));
            panel.add(toolbar, BorderLayout.NORTH);

            SDToolButtonFactory toolButtonFactory = new SDToolButtonFactory(model);
            toolbar.add(toolButtonFactory.createDrawTools());
            toolbar.add(toolButtonFactory.createSCTools());

            GPanel gpanel = new GPanel(model);
            model.setView(panel);
            SDEditorControl control = new SDEditorControl(model);
            gpanel.addMouseListener(control);
            gpanel.addMouseMotionListener(control);
            panel.add(gpanel);

            tabbedPane.addTab("Drawing", panel);
        }

        JPanel colorPanel = new JPanel(false);
        tabbedPane.addTab("Color", colorPanel);

        JPanel analyzePanel = new JPanel(false);
        tabbedPane.addTab("Analyze", analyzePanel);
}
```

```
SDView(Container topPane) {
        JTabbedPane tabbedPane = new JTabbedPane();
        topPane.add(tabbedPane);

        {
            JPanel panel = new JPanel(false);
            panel.setLayout(new BorderLayout());

            JPanel toolbar = new JPanel();
            toolbar.setLayout(new FlowLayout(FlowLayout.LEFT));
            panel.add(toolbar, BorderLayout.NORTH);

            SDToolButtonFactory toolButtonFactory = new SDToolButtonFactory(model);
            toolbar.add(toolButtonFactory.createDrawTools());
            toolbar.add(toolButtonFactory.createSCTools());

            GPanel gpanel = new GPanel(model);
            model.setView(panel);
            SDEditorControl control = new SDEditorControl(model);
            gpanel.addMouseListener(control);
            gpanel.addMouseMotionListener(control);
            panel.add(gpanel);

            tabbedPane.addTab("Drawing", panel);
        }

        JPanel colorPanel = new JPanel(false);
        tabbedPane.addTab("Color", colorPanel);

        JPanel analyzePanel = new JPanel(false);
        tabbedPane.addTab("Analyze", analyzePanel);
}
```

スコープを限定する技術2

while < for < for (:)

```
for (Element e : c) {
    doSomething(e);
}
```

```
for (Iterator iter = c.iterator(); iter.hasNext(); ) {
    doSomething((Element)iter.next());
}
```

```
Iterator<Element> iter = c.iterator();
while (iter.hasNext()) {
    doSomething((Element)iter.next());
}
```

Prefer primitive types to boxed primitives

Primitive vs Boxed

Primitive Type	Boxed class
boolean	Boolean
byte	Byte
short	Short
int	Integer
float	Float
double	Double

Primitive vs Boxed

- Primitive (原子的な)
  - 計算機にとって基本的 ⇒
    ハードウェアで直接扱うことができる ⇒
    レジスタに保存できる
- Boxed (箱に納まった)
  - （レジスタではなく）メモリにオブジェクトとして保存した値

Primitive ⇔ Boxed

- → (Boxing)
  - Integer n = new Integer(5)
- ← (Unboxing)
  - int v = (int)n;

Auto-boxing/unboxing

- Java 1.5 以来、boxing/unboxing は自動化されるようになった。
  - Boxing
    - Integer n = 5;
  - Unboxing
    - int v = n;

Boxedを避ける理由

- 非効率
  - 大量のメモリを消費する
- 危なっかしい
  - オブジェクトであることを忘れがち

効率の違い

```
package lecture03;

public class IncrementBenchmark {
    private void run() {
        long start_ms = System.currentTimeMillis();
        {
            long sum = 0L;
            for (long i = 0L; i < Integer.MAX_VALUE; i++) sum += i;
            long end_ms = System.currentTimeMillis();
            System.out.printf("%d (%dms)\n", sum, end_ms - start_ms);
            start_ms = end_ms;
        }

        {
            Long sum = 0L;
            for (Long i = 0L; i < Integer.MAX_VALUE; i++) sum += i;
            long end_ms = System.currentTimeMillis();
            System.out.printf("%d (%dms)\n", sum, end_ms - start_ms);
            start_ms = end_ms;
        }
    }

    public static void main(String _ []) {
        new IncrementBenchmark().run();
    }
}
```

効率の違い

```
package lecture03;

public class IncrementBenchmark {
    private void run() {
        long start_ms = System.currentTimeMillis();
        {
            long sum = 0L;
            for (long i = 0L; i < Integer.MAX_VALUE; i++) sum += i;
            long end_ms = System.currentTimeMillis();
            System.out.printf("%d (%dms)\n", sum, end_ms - start_ms);
            start_ms = end_ms;
        }

        {
            Long sum = 0L;
            for (Long i = 0L; i < Integer.MAX_VALUE; i++) sum += i;
            long end_ms = System.currentTimeMillis();
            System.out.printf("%d (%dms)\n", sum, end_ms - start_ms);
            start_ms = end_ms;
        }
    }

    public static void main(String _ []) {
        new IncrementBenchmark().run();
    }
}
```

```
2305843005992468481 (1826ms)
2305843005992468481 (51459ms)
```

効率の違い

```
package lecture03;

public class IncrementBenchmark {
    private void run() {
        long start_ms = System.currentTimeMillis();
        {
            long sum = 0L;
            for (long i = 0L; i < Integer.MAX_VALUE; i++) sum += i;
            long end_ms = System.currentTimeMillis();
            System.out.printf("%d (%dms)\n", sum, end_ms - start_ms);
            start_ms = end_ms;
        }

        {
            Long sum = 0L;
            for (Long i = 0L; i < Integer.MAX_VALUE; i++) sum += i;
            long end_ms = System.currentTimeMillis();
            System.out.printf("%d (%dms)\n", sum, end_ms - start_ms);
            start_ms = end_ms;
        }
    }

    public static void main(String _ []) {
        new IncrementBenchmark().run();
    }
}
```

1.8秒

51秒

```
2305843005992468481 (1826ms)
2305843005992468481 (51459ms)
```

効率の違い

1.8秒

28倍の差

51秒

```
package lecture03;

public class IncrementBenchmark {
    private void run() {
        long start_ms = System.currentTimeMillis();
        {
            long sum = 0L;
            for (long i = 0L; i < Integer.MAX_VALUE; i++) sum += i;
            long end_ms = System.currentTimeMillis();
            System.out.printf("%d (%dms)\n", sum, end_ms - start_ms);
            start_ms = end_ms;
        }

        {
            Long sum = 0L;
            for (Long i = 0L; i < Integer.MAX_VALUE; i++) sum += i;
            long end_ms = System.currentTimeMillis();
            System.out.printf("%d (%dms)\n", sum, end_ms - start_ms);
            start_ms = end_ms;
        }
    }

    public static void main(String _ []) {
        new IncrementBenchmark().run();
    }
}
```

```
2305843005992468481 (1826ms)
2305843005992468481 (51459ms)
```

一見、問題ないが。。。

```
// Broken comparator - can you spot the flaw?
Comparator<Integer> naturalOrder = new Comparator<Integer>() {
    public int compare(Integer first, Integer second) {
        return first < second ? -1 : (first == second ? 0 : 1);
    }
};
```

```java
package lecture03;

public class ComparatorBug {
    public int compare1(Integer x1, Integer x2) {
        int n1 = x1, n2 = x2;
        return n1 < n2 ? -1 : n1 == n2 ? 0 : 1;
    }

    public int compare2(Integer x1, Integer x2) {
        return x1 < x2 ? -1 : x1 == x2 ? 0 : 1;
    }

    private void run() {
        Integer two = 2;
        for (Integer i = 1; i <= 3; i++) {
            Integer n = new Integer(i);
            System.out.printf("compare1(2, %d) -> %d\n", n, compare1(two, n));
            System.out.printf("compare2(2, %d) -> %d\n", n, compare2(two, n));
        }
    }

    public static void main(String[] _) {
        new ComparatorBug().run();
    }
}
```

```java
package lecture03;

public class ComparatorBug {
    public int compare1(Integer x1, Integer x2) {
        int n1 = x1, n2 = x2;
        return n1 < n2 ? -1 : n1 == n2 ? 0 : 1;
    }

    public int compare2(Integer x1, Integer x2) {
        return x1 < x2 ? -1 : x1 == x2 ? 0 : 1;
    }

    private void run() {
        Integer two = 2;
        for (Integer i = 1; i <= 3; i++) {
            Integer n = new Integer(i);
            System.out.printf("compare1(2, %d) -> %d\n", n, compare1(two, n));
            System.out.printf("compare2(2, %d) -> %d\n", n, compare2(two, n));
        }
    }

    public static void main(String[] _) {
        new ComparatorBug().run();
    }
}
```

```
public class Unbelievable {
    static Integer i;

    public static void main(String[] args) {
        if (i == 42)
            System.out.println("Unbelievable");
    }
}
```

```java
package lecture03;

public class BoxedTest {
    static int vi;
    private void test1() {
        System.out.printf("test1: vi %s 42\n", vi == 42 ? "==" : "!=");
    }

    static Integer i;
    private void test2() {
        System.out.printf("test1: i %s 42\n", i == 42 ? "==" : "!=");
    }

    private void run() { test1(); test2(); }

    public static void main(String[] _) {
        new BoxedTest().run();
    }
}
```

```
package lecture03;

public class BoxedTest {
    static int vi;
    private void test1() {
        System.out.printf("test1: vi %s 42\n", vi == 42 ? "==" : "!=");
    }

    static Integer i;
    private void test2() {
        System.out.printf("test1: i %s 42\n", i == 42 ? "==" : "!=");
    }

    private void run() { test1(); test2(); }

    public static void main(String[] _) {
        new BoxedTest().run();
    }
}
```

```
package lecture03;

public class BoxedTest {
    static int vi;
    private void test1() {
        System.out.printf("test1: vi %s 42\n", vi == 42 ? "==" : "!=");
    }

    static Integer i;
    private void test2() {
        System.out.printf("test1: i %s 42\n", i == 42 ? "==" : "!=");
    }
```

```
test1: vi != 42
Exception in thread "main" java.lang.NullPointerException
        at lecture03.BoxedTest.test2(BoxedTest.java:12)
        at lecture03.BoxedTest.run(BoxedTest.java:17)
        at lecture03.BoxedTest.main(BoxedTest.java:21)
```

```
}
```

次回

- オブジェクト生成の技
- Item 1:Consider ***static factory methods*** instead of constructors
- Item 2:Consider a ***builder*** when faced with many constructor parameters