CS 520
Theory and Practice of Software Engineering
Fall 2017

Best and worst programming practices

September 12, 2017
## Logistics

### Course material

<table>
<thead>
<tr>
<th>Date</th>
<th>Topic</th>
<th>Material</th>
<th>Homework</th>
</tr>
</thead>
<tbody>
<tr>
<td>09/05/2017</td>
<td>Course introduction</td>
<td>Slides, Slides (4pages)</td>
<td></td>
</tr>
<tr>
<td>09/07/2017</td>
<td>Software architecture and design/UML crash course</td>
<td>Slides, Slides (4pages)</td>
<td></td>
</tr>
<tr>
<td>09/12/2017</td>
<td>Best and worst programming practices</td>
<td></td>
<td></td>
</tr>
<tr>
<td>09/14/2017</td>
<td>OO design principles</td>
<td></td>
<td></td>
</tr>
<tr>
<td>09/19/2017</td>
<td>Version control systems/Git tutorial</td>
<td></td>
<td></td>
</tr>
<tr>
<td>09/21/2017</td>
<td><strong>In-class exercise:</strong> Advanced uses of git</td>
<td></td>
<td></td>
</tr>
<tr>
<td>09/26/2017</td>
<td>OO design patterns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>09/28/2017</td>
<td>OO design patterns</td>
<td></td>
<td><strong>HW 1 online</strong></td>
</tr>
<tr>
<td>10/03/2017</td>
<td>Functional programming in Java 8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10/05/2017</td>
<td>Intro to empirical software engineering</td>
<td></td>
<td><strong>PR 1 is due (10am)</strong></td>
</tr>
<tr>
<td>10/10/2017</td>
<td><strong>No class:</strong> Monday class schedule will be followed!</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10/12/2017</td>
<td>Experimental design and validity</td>
<td></td>
<td><strong>PR 2 is due (10am)</strong></td>
</tr>
<tr>
<td>10/17/2017</td>
<td>Intro to program analysis and software testing</td>
<td></td>
<td><strong>HW 1 is due</strong></td>
</tr>
<tr>
<td>10/19/2017</td>
<td><strong>In-class exercise:</strong> Software testing</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Recap: software architecture vs. design

What’s the difference?
Recap: software architecture examples

- **Pipe and filter**
  
  ```bash
  grep CS520 grades.csv | cut -f 1 -d ',' | sort | uniq -c
  ```

  ```
  A,CS320,Joe
  B,CS520,Jane
  ...
  ```
  
  2 A
  1 B
  ...

- **N-tier / client-server**

- **MVC (Model-View-Controller)**
Recap: software architecture and design goals

Architecture and design goals
- Lower complexity: separation of concerns, well defined interfaces
- Simplify communication
- Allow effort estimation and progress monitoring
Today

- A little quiz on best/worst programming practices.
Quiz: setup and goals

● 6-12 teams
● 6 code snippets
● 2 rounds
  ○ First round
    ■ For each code snippet, decide whether it represents good or bad practice.
    ■ Goal: discuss and reach consensus on good or bad practice.
  ○ Second round (known solutions)
    ■ For each code snippet, try to understand why it is good or bad practice.
    ■ Goal: come up with one or more explanations or a counter argument.
Round 1: good or bad?
public File[] getAllLogs(Directory dir) {
    if (dir == null || !dir.exists() || dir.isEmpty()) {
        return null;
    } else {
        int numLogs = ... // determine number of log files
        File[] allLogs = new File[numLogs];
        for (int i=0; i<numLogs; ++i) {
            allLogs[i] = ... // populate the array
        }
        return allLogs;
    }
}
public void addStudent(Student student, String course) {
    if (course.equals("CS520")) {
        cs520Students.add(student);
    }
    allStudents.add(student)
}
public enum PaymentType {DEBIT, CREDIT}

public void doTransaction(double amount, PaymentType payType) {
    switch (payType) {
    case DEBIT:
        // process debit card
        break;
    case CREDIT:
        // process credit card
        break;
    default:
        throw new IllegalArgumentException("Unexpected payment type");
    }
}
Snippet 4: good or bad?

```java
public int getAbsMax(int x, int y) {
    if (x<0) {
        x = -x;
    }
    if (y<0) {
        y = -y;
    }
    return Math.max(x, y);
}
```
public class ArrayList<E> {
    public E remove(int index) {
        ...
    }
    public boolean remove(Object o) {
        ...
    }
    ...
}
public class Point {
    private final int x;
    private final int y;

    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public int getX() {
        return this.x;
    }

    public int getY() {
        return this.y;
    }
}
Solutions

- Snippet 1: bad
- Snippet 2: bad
- Snippet 3: good
- Snippet 4: bad
- Snippet 5: bad
- Snippet 6: good
Round 2: why is it good or bad?
Snippet 1: this is bad! why?

```java
public File[] getAllLogs(Directory dir) {
    if (dir == null || !dir.exists() || dir.isEmpty()) {
        return null;
    } else {
        int numLogs = ... // determine number of log files
        File[] allLogs = new File[numLogs];
        for (int i=0; i<numLogs; ++i) {
            allLogs[i] = ... // populate the array
        }
        return allLogs;
    }
}
```
Snippet 1: this is bad! why?

```java
public File[] getAllLogs(Directory dir) {
    if (dir == null || !dir.exists() || dir.isEmpty()) {
        return null;
    } else {
        int numLogs = ... // determine number of log files
        File[] allLogs = new File[numLogs];
        for (int i=0; i<numLogs; ++i) {
            allLogs[i] = ... // populate the array
        }
        return allLogs;
    }
}
```

Don't return null; return an empty array instead.

```java
File[] files = getAllLogs();
for (File f : files) {
    ...
}
```
Snippet 2: short but bad! why?

```java
public void addStudent(Student student, String course) {
    if (course.equals("CS520")) {
        cs520Students.add(student);
    }
    allStudents.add(student)
}
```
public void addStudent(Student student, String course) {
    if (course.equals("CS520")) {
        cs520Students.add(student);
    }
    allStudents.add(student)
}
public enum PaymentType {DEBIT, CREDIT}

public void doTransaction(double amount, PaymentType payType) {
    switch (payType) {
        case DEBIT:
            ... // process debit card
            break;
        case CREDIT:
            ... // process credit card
            break;
        default:
            throw new IllegalArgumentException("Unexpected payment type");
    }
}
public enum PaymentType {DEBIT, CREDIT}

public void doTransaction(double amount, PaymentType payType) {
    switch (payType) {
        case DEBIT:
            // process debit card
            break;
        case CREDIT:
            // process credit card
            break;
        default:
            throw new IllegalArgumentException("Unexpected payment type");
    }
}

Type safety using an enum; throws an exception for unexpected cases (e.g., future extensions of PaymentType).
Snippet 4: also bad! huh?

```java
public int getAbsMax(int x, int y) {
    if (x<0) {
        x = -x;
    }
    if (y<0) {
        y = -y;
    }
    return Math.max(x, y);
}
```
Snippet 4: also bad! huh?

```java
public int getAbsMax(int x, int y) {
    if (x < 0) {
        x = -x;
    }
    if (y < 0) {
        y = -y;
    }
    return Math.max(x, y);
}
```

Method parameters should be final; use local variables to sanitize inputs.
Snippet 5: Java API, but still bad! why?

```java
public class ArrayList<E> {
    public E remove(int index) {
        ...
    }
    public boolean remove(Object o) {
        ...
    }
    ...
}
```
Snippet 5: Java API, but still bad! why?

```java
public class ArrayList<E> {
    public E remove(int index) {
        ...
    }
    public boolean remove(Object o) {
        ...
    }
    ...
}

ArrayList<String> l = new ArrayList<>();
Integer index = new Integer(1);
l.remove(index);
```

Avoid method overloading, which is statically resolved. Autoboxing/unboxing adds additional confusion.
public class Point {
    private final int x;
    private final int y;

    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public int getX() {
        return this.x;
    }

    public int getY() {
        return this.y;
    }
}
Snippet 6: this is good, but why?

```java
public class Point {
    private final int x;
    private final int y;

    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public int getX() {
        return this.x;
    }

    public int getY() {
        return this.y;
    }
}
```

Good encapsulation; immutable object.