MASSEY UNIVERSITY
TE KUNENGA KI PŪREHUROA
UNIVERSITY OF NEW ZEALAND

Software Design and Construction

159.251

DRY and the Evils of Duplication

Amjed Tahir
a.tahir@massey.ac.nz

Original author: Jens Dietrich

References

[PP] Andrew Hunt and David Thomas:
The Pragmatic Programmer: From Journeyman to Master.
Addison-Wesley, Oct 1999.

[CC] Robert Martin:
Clean Code: A Handbook of Agile Software Craftsmanship.
Prentice Hall 2009.

[EJ] Joshua Bloch:
Effective Java Second Edition.
Sun Micro 2008.

The Evils of Duplication

- programs require constant **maintenance**

- this begins during development - code is continuously changed as the understanding of the program increases
- once the software is deployed, there is continuous **change pressure**:
  - bug fixes
  - adding new features
  - adapting to changing environments (OS and VM versions, regulations, other systems, etc)

The Evils of Duplication (ctd)

- Copy & Paste (C&P) makes it easy to **duplicate artefacts** quickly
  artefacts are not only code, but also models (like UML), configuration files, comments, documentation etc

- if an artefact changes, all of its copies must be detected and must be consistently changed as well

- **this makes systems error-prone, and maintenance expensive**

Example of code (C&P) duplication

Apache Commons Lang3 (2.6)

Class something

```
 if (array == null) {
            return;
        }
        if (startIndexInclusive >= array.length - 1 ||
endIndexExclusive <= 0) {
            return;
        }
        if (startIndexInclusive < 0) {
            startIndexInclusive = 0;
        }
        if (endIndexExclusive >= array.length) {
            endIndexExclusive = array.length;
        }
        int n = endIndexExclusive -
startIndexInclusive;
        if (n <= 1) {
            return;
        }
```

Class SomethingElse

```
 if (array == null) {
            return;
        }
        if (startIndexInclusive >= array.length - 1 ||
endIndexExclusive <= 0) {
            return;
        }
        if (startIndexInclusive < 0) {
            startIndexInclusive = 0;
        }
        if (endIndexExclusive >= array.length) {
            endIndexExclusive = array.length;
        }
        int n = endIndexExclusive -
startIndexInclusive;
        if (n <= 1) {
            return;
        }
        offset %= n;
        if (offset < 0) {
            offset += n;}
```

The DRY Principle

Don't **R**epeat **Y**ourself

Every piece of knowledge must have a single, unambiguous, authoritative representation within the system. [PP]

Where does Duplication Come From?

(from [PP])

- imposed duplication

  No choice but to duplicate! You must duplicate due to a particular reason.

- inadvertent duplication

  you duplicate without realising it! Very common!

- impatient duplication

  Doing it the dirty way! You just feel lazy and decide to duplicate because it feels easy!

- inter-developer duplication

  Multiple people on the same project/team duplicates a piece of code

Imposed Duplication

- multiple representation of information

    Example: comments duplicating information in code

- duplication caused by programming language (example: headers, CORBA interfaces)

Multiple Representation (ctd)

- many application have a layered model
- layers (aka tiers):
  - **User Interface (UI) layer** - forms to edit data, windows (desktop) or web-based
  - **Domain layer** - objects representing data structures
  - **Persistency layer** - data representation suitable for persistency, such as database tables or structured files

Multiple Representation Examples

- UI layer: a **form** to edit employees with **fields** to edit name, first name, date of birth, tax number, + 20 more, and a link to a form to edit the address

- Domain layer: an **Employee class** with name, firstName, dateOfBirth + 20 more **properties**, and another property of the type Address

- Persistency layer: a **table EMPLOYEE** with **columns** name, first_name, dob + 20 more, and a foreign key reference to a table ADDRESS

Avoiding Multiple Representation

- the representation can be mapped: the mapping rules are well-understood

- but! mapping rules can be complex, in particular when mapping objects to relational databases - this is known as **Object-Relational Mapping (ORM)**

- mapping rules can be used to write code generators

Successful Code Generators

- ORM frameworks such as Hibernate and JOOQ can create database schemas from class definitions

- database schemas = table definitions, i.e. DDL statements such as CREATE TABLE are generated

- RubyOnRails is a successful framework to create database schemas as well as user interfaces (forms) from class definitions

Which Way?

- it is sometimes hard to decide where to start
- i.e., what is the primary representation of information?
- an issue to consider is maintenance - where will changes occur?
- some tools can synchronise different representations
- this is also called **roundtrip engineering**
- roundtrip engineering = forward engineering + reverse engineering

Case Study: JAXB – Background

Java Architecture for XML Binding (JAXB)

- XML is a popular format to represent structured data
- It can be used to encode/decode objects to/from streams (you will learn more about this later!)
- main applications:
  - persistency (files: office documents, configuration files, ..)
  - networking (web services: SOAP)
- structure of XML documents is described by a schema
- popular schema languages:
  - XMLSchema (XSD)
  - Document Type Definition (DTD)
- source code: https://bitbucket.org/jensdietrich/oop-examples/src/1.0/jaxb/

Case Study: JAXB - Problem

How to map java objects/classes to XML?

- structure of Java (classes) will be similar to structure of XML document: types, relationships and names
- solution strategies:
    - parse text directly (tokenizing, regex (regular expression) etc)

      difficult and error-prone, problems with details (escaping characters etc)
    - use Java XML API –

      better, but still requires a lot of manual work
    - **generate matching classes + parser from schema (XSD, DTD, etc)**

Case Study: JAXB - XSD Example

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
        <xs:complexType name="participant">
                <xs:sequence>
                        <xs:element name="email_address" type="xs:string"/>
                        <xs:element name="display_name" type="xs:string" minOccurs="0"/>
                </xs:sequence>
        </xs:complexType>
        <xs:element name="email">
                <xs:complexType>
                        <xs:sequence minOccurs="0">
                                <xs:element name="to" type="participant" maxOccurs="32"/>
                                <xs:element name="cc" type="participant" minOccurs="0" maxOccurs="32"/>
                                <xs:element name="bcc" type="participant" minOccurs="0" maxOccurs="32"/>
                                <xs:element name="subject" type="xs:string" minOccurs="0"/>
                                <xs:element name="body" type="xs:string" minOccurs="0"/>
                        </xs:sequence>
                        <xs:attribute name="id" type="xs:NMTOKEN" use="required"/>
                </xs:complexType>
        </xs:element>
</xs:schema>
```

defines concepts email and participant

Case Study: JAXB - Instance example

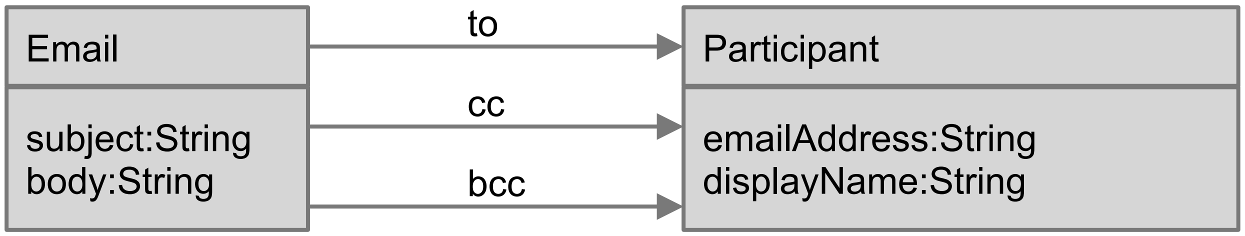
```
<?xml version="1.0" encoding="UTF-8"?>
<email id="id1" .. xsi:noNamespaceSchemaLocation="email.xsd">
        <to>
                <email_address>jens@massey.ac.nz</email_addr
                <display_name>Jens Dietrich</display_name>
        </to>
        <cc>
                <email_address>students159251_2012@massey.ac.nz</email_address>
                <display_name>159.251 2012 student list</display_name>
        </cc>
        <subject>update</subject>
        <body>this lecture notes have been updated</body>

</email>
```

email instance

participant instance

Case Study: JAXB - Java Representation



this is a UML class diagram (you will learn more about this this year and next year (158225))
http://www.objectmentor.com/resources/articles/umlClassDiagrams.pdf

Case Study: JAXB - Generating

```
xjc -d src -p nz.ac.massey.cs.sdc.jaxb email.xsd
```

- jaxb is in the JDK bin folder
- -p option - package name
- -d option - destination folder
- generates a class per complex type:
  - `nz.ac.massey.cs.sdc.jaxb.Email`
  - `nz.ac.massey.cs.sdc.jaxb.Participant`
- generates helper class
  `nz.ac.massey.cs.sdc.jaxb.ObjectFactory`
- terminology:
  - marshall = serialise, save object to stream
  - unmarshall = deserialise, parse object from stream

Case Study: JAXB - Parsing

```
@Test
public void test() throws Exception {
       JAXBContext jc =
       JAXBContext.newInstance( "nz.ac.massey.cs.sdc.jaxb" );
       Unmarshaller parser = jc.createUnmarshaller();
       File file = new File("email1.xml");
       Email mail = (Email) parser.unmarshal(file);

       assertEquals(1,mail.getTo().size());
       Participant to = mail.getTo().get(0);
       assertEquals("jens@massey.ac.nz",to.getEmailAddress());
       assertEquals("Jens Dietrich",to.getDisplayName());

       ... // more asserts
```

Case Study: JAXB - Conclusion

- jaxb works great for simple schemas
- jaxb can be automated - generate classes while software is build
- does not work well if change occurs in OO model first
- conceptual mismatch: xml elements can have children of different types (sequence of several different complex types), but Java does not support "union types"

Domain-Driven Design

- domain-driven design (DDD) refers to an approach where the focus is on developing the domain model
- the domain model contains representations of concepts needed to describe the domain, and the main application logic
- user interface and persistency layer (usually database) can be generated from the domain model
- this approach is sometimes also called the **Naked Object pattern**
- an example for DDD is the (open source) Apache Isis project and MetaWidget

Documentation and Code

- people often over-document: comments duplicate code
- this if often taught at universities ("document as much as possible"), but often leads to redundancies
- better: **self-documenting code**

Redundant Comment Example

```
/**
 * Sets the name, may throw an exception. This method
 * does not return a value. Can be used by other classes.
 * @param name a string representing the new name
 */
public void setName(String name) throws Exception {
      this.name = name;
}
```

Redundant Comment Example (ctd)

```
/**
 * Sets the name, may throw an exception. This method
 * does not return a value. Can be used by other classes.
 * @param name a string representing the new name
 */
public void setName(String name) throws Exception {
       this.name = name;
}
```

- these comments do not convey meaningful information
- these comments are not useful, and also harmful – if changes are made and these comments are not updated, the user will be confused

Language Issues

- sometimes, duplication is imposed by issues in the language
- often this happens when frameworks require that interfaces must be separated from classes

- in Java, this is the case with CORBA - IDL interfaces replicate structural information of classes
- another example is setters and getters

CORBA : Common Object Request Broker Architecture
IDL: Interface Description Language

Setters and Getters in Java

- Java mandates that properties are implemented as follows:
  - a public getter (aka accessor):
    ```
    public String getName()
    ```
  - a public setter (aka mutator):
    ```
    public void setName(String name)
    ```
  - this is usually combined with a private field, e.g.:
    ```
    private String name
    ```

- this means that the following information is duplicated:
  - property name
  - property type

Alternative getters+setters: C#

```
private string name;
public string Name {
   get { return name; }
   set { name = value; }
}
```

or (from C# 3.0 or better):

```
public string Name { get; set; }
```

- more compact definition
- name and type not replicated in setter/getter

see also https://msdn.microsoft.com/en-us/library/bb384054.aspx

Alternative getters+setters: Ruby

```
class Person
  def initialize(pid)
       @name=""
              @firstName=""
  end
  attr_accessor :name,:firstName
end
```

- all of the instance variables completely private to the class (access through accessor methods)
- `attr_accessor` defines setters/getters
- this is called **meta-programming**
- setter and getter syntax looks like direct field access:
  - in `person.name="Tom"`, the setter is the method **`name=()`**

Dealing with Setters/Getters in Java

- maintenance is supported by compiler: if name or type of property is changed, the compiler enforces consistency (Eclipse: errors markers occur)
- E.g., if the field type is changed, the return type of getter must (usually) also be changed - otherwise a compiler error occurs
- IDEs like Eclipse have built-in code generators to generate setters+getters and other methods that requires access to fields (equals, hashcode, toString)
- setters and getters are important - they have meaning, and this is used by tools such as UI builders
- to be discussed later .. "programming by convention"

Inadvertent Duplication

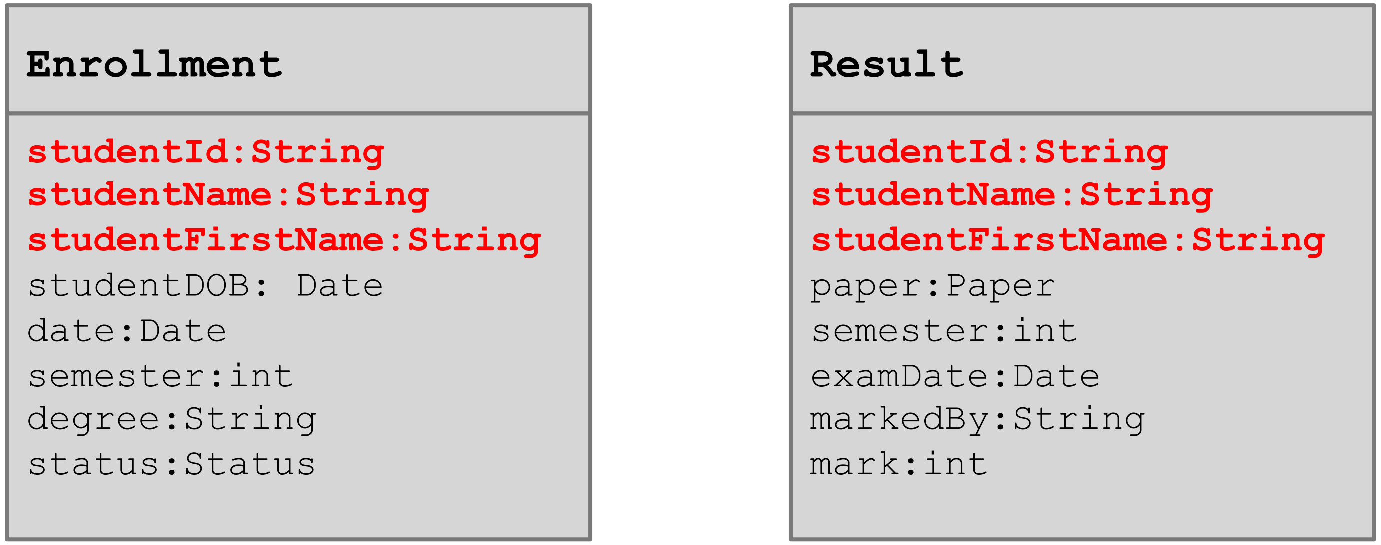
- duplication by design
- design fails to spot duplication, and creates multiple representation of the same objects

Inadvertent Duplication Example

- consider the following model developed to represent several aspects of the University processes:
    - degree enrollment (ENROL)
    - result processing (RP)

- assume we have created the following classes from requirements
- the same information about students is then represented twice

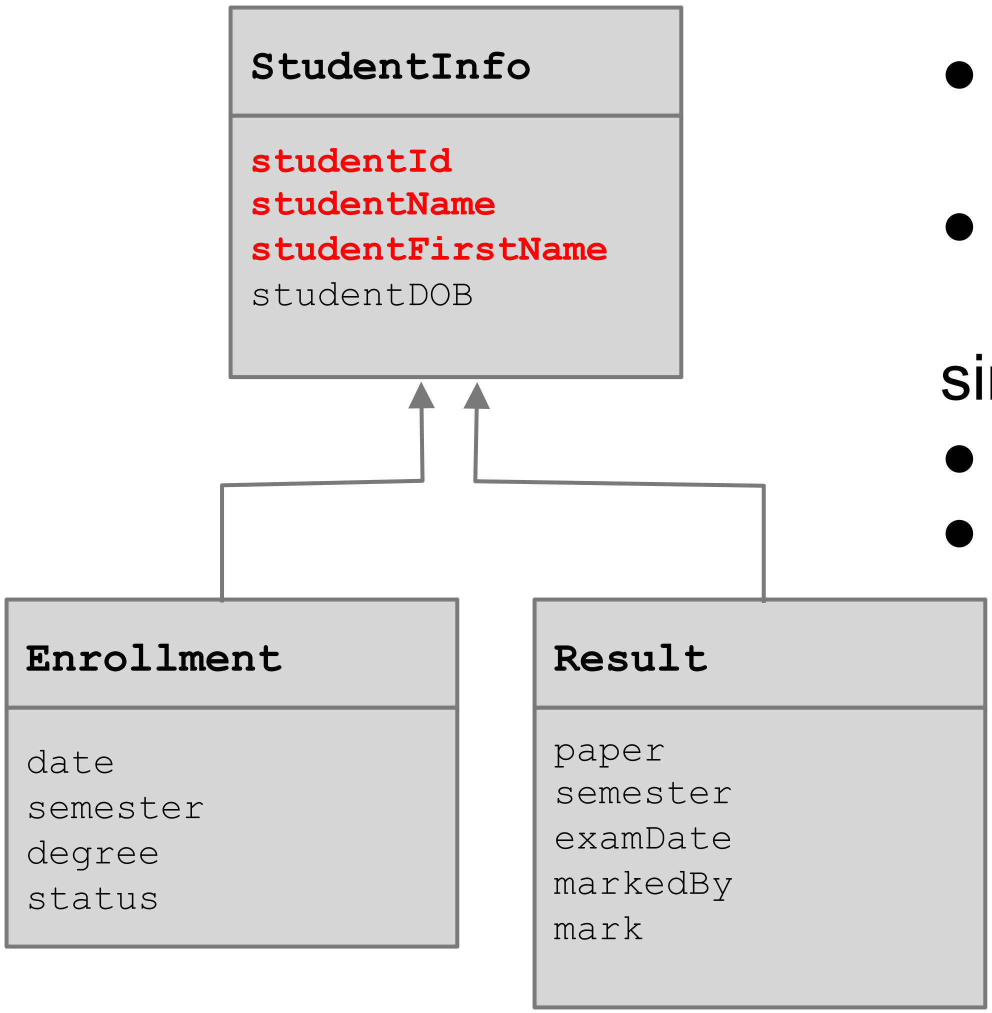
Inadvertent Duplication Example (ctd)



```
Enrollment
----------
studentId:String
studentName:String
studentFirstName:String
studentDOB: Date
date:Date
semester:int
degree:String
status:Status
```

```
Result
------
studentId:String
studentName:String
studentFirstName:String
paper:Paper
semester:int
examDate:Date
markedBy:String
mark:int
```

Solution 1: Use Inheritance

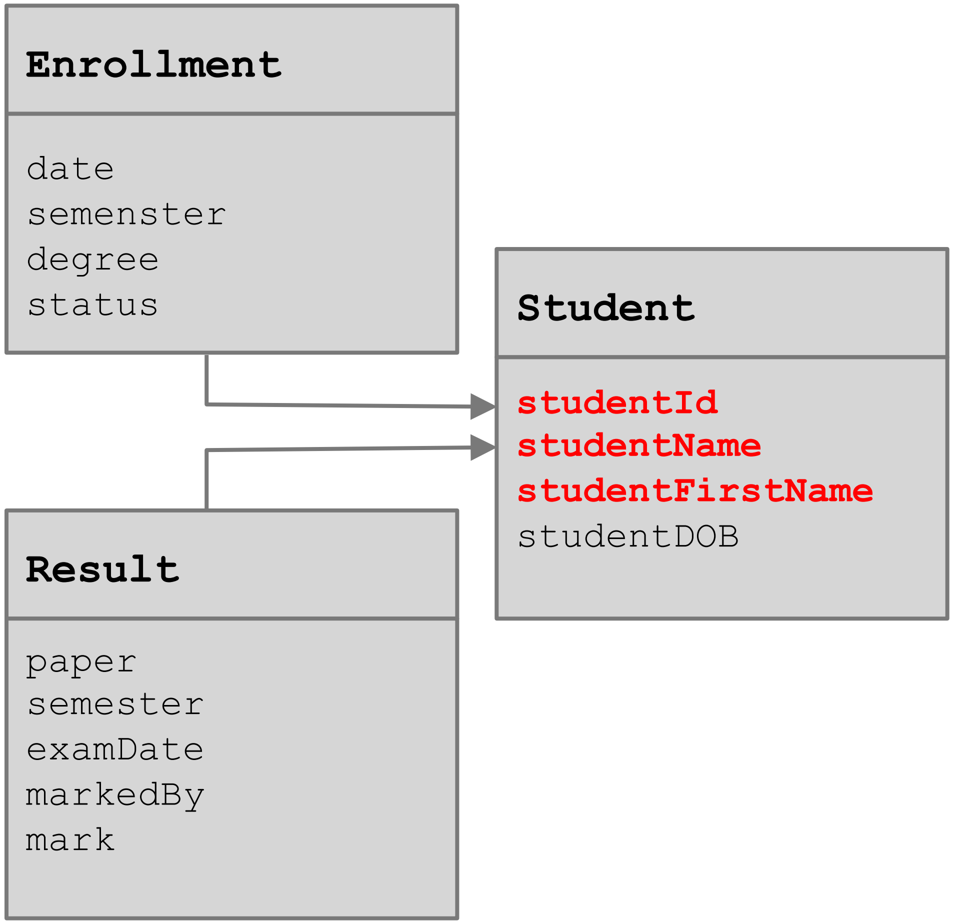


- create a common superclass
- **pull up** common properties

simple, but:

- superclass not intuitive
- can only use one (direct) superclass - "one shot only"

Solution 2: Use Composition



- create class
- create associations with this class
- in Java: properties of type `Student` in `Enrollment` and `Result`
- more intuitive and flexible
- *inheritance vs composition will be discussed later in detail!*

Duplication in (Relational) DBs (ctd)

- the evils of duplication are also studied in database design (starting with the work of Edgar Codd around 71)
- the problem is **update anomalies** - when duplicates exit, information can become inconsistent during updates
- this leads to **database normalisation** - data representation where duplication has been removed
- the key idea is to replace copies with references

Duplication in (Relational) Data Bases

Employees' Skills

Employee	Skill	Current Work Location
Jones	Typing	114 Main Street
Jones	Shorthand	114 Main Street
Jones	Whittling	114 Main Street
Bravo	Light Cleaning	73 Industrial Way
Ellis	Alchemy	73 Industrial Way
Ellis	Flying	73 Industrial Way
Harrison	Light Cleaning	73 Industrial Way

- work location for employees is duplicated
- when one record is updated, and others are forgotten, an **update anomaly** occurs

Employees

Employee	Current Work Location
Jones	114 Main Street
Bravo	73 Industrial Way
Ellis	73 Industrial Way
Harrison	73 Industrial Way

Employees' Skills

Employee	Skill
Jones	Typing
Jones	Shorthand
Jones	Whittling
Bravo	Light Cleaning
Ellis	Alchemy
Ellis	Flying
Harrison	Light Cleaning

- this is solved by splitting the table
- rows from different tables can be linked (cross-referenced) using foreign keys
- this is called second **normal form** (NF2)

example from http://en.wikipedia.org/wiki/Second_normal_form

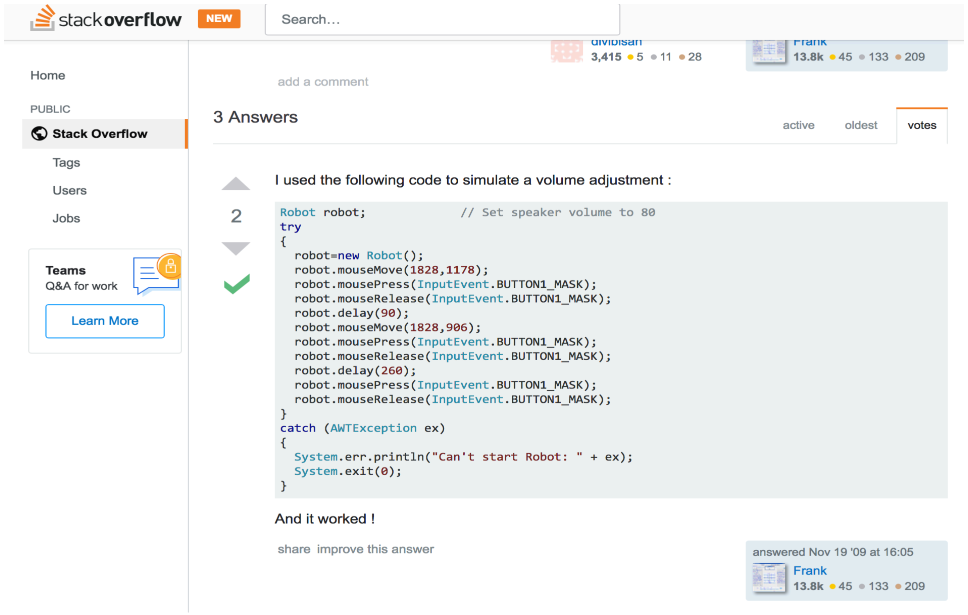
Duplication in XML Schema

```
...
<xs:complexType name="participant">
        <xs:sequence>
                <xs:element name="email_address" type="xs:string"/>
                <xs:element name="display_name" type="xs:string" minOccurs="0"/>
        </xs:sequence>
</xs:complexType>
<xs:element name="email">
        <xs:complexType>
                <xs:sequence minOccurs="0">
                        <xs:element name="to" type="participant" maxOccurs="32"/>
                        <xs:element name="cc" type="participant" minOccurs="0" maxOccurs="32"/>
                        <xs:element name="bcc" type="participant" minOccurs="0" maxOccurs="32"/>
                        <xs:element name="subject" type="xs:string" minOccurs="0"/>
                        ...
```

- The XML Schema used in an earlier example avoided duplication
- `to`,`cc` and `bcc` have the same internal structure ("shared content model")
- this is expressed by creating a complex type participant **referenced** by those elements

Impatient Duplication

- take advantage of convenience of copy & paste
- "short cuts make for long delays"
- special case: copy and paste code snippets from the web



add a comment

3 Answers

I used the following code to simulate a volume adjustment :

```
Robot robot;              // Set speaker volume to 80
try
{
  robot=new Robot();
  robot.mouseMove(1828,1178);
  robot.mousePress(InputEvent.BUTTON1_MASK);
  robot.mouseRelease(InputEvent.BUTTON1_MASK);
  robot.delay(90);
  robot.mouseMove(1828,906);
  robot.mousePress(InputEvent.BUTTON1_MASK);
  robot.mouseRelease(InputEvent.BUTTON1_MASK);
  robot.delay(260);
  robot.mousePress(InputEvent.BUTTON1_MASK);
  robot.mouseRelease(InputEvent.BUTTON1_MASK);
}
catch (AWTException ex)
{
  System.err.println("Can't start Robot: " + ex);
  System.exit(0);
}
```

And it worked !

share improve this answer

answered Nov 19 '09 at 16:05
Frank
13.8k ●45 ●133 ●209

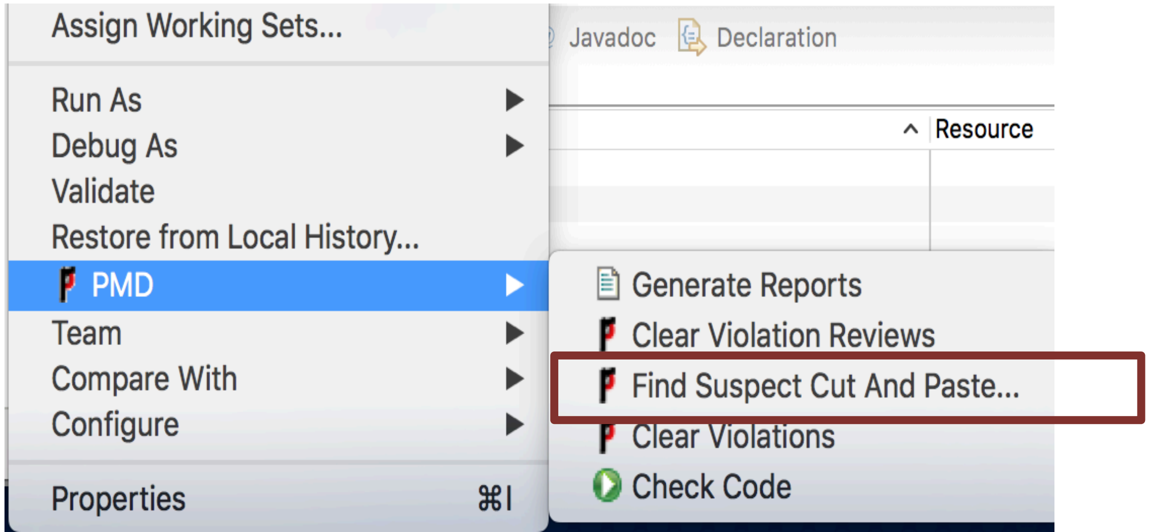
- make sure that only one master copy exists that can be maintained easily

Interdeveloper Duplication

- lack of communication between developers (and entire projects) leads to duplication or great blocks of functionality
- this is not "copy & paste" duplication, but the existence of several modules expressing the same problem
- common examples: client module, student management, authentication
- solutions:
  - improved communication between developers and stakeholders, e.g., use of social networking (forums etc)
  - encourage & reward reuse

Tooling: PMD-CPD

- More on PMD on topic 11
- it is a former DARPA, now open source project
- PMD includes a **copy-and-paste detector** (CPD)
- to run CPD in Eclipse, click on project then **PMD > Find cut and paste ..**
- PMD CPD is based on string matching
- CPD report for JDK1.4: http://pmd.sourceforge.net/cpdresults.txt



Plaggie

- plaggie is a tool to find structural similarity in programs
- not text, but the structure is compared
- programs have a unique tree structure, the so-called Abstract Syntax Tree (AST)
- PMD also has a function to compute the AST
- even if variables in a program are renamed, the structure remains identical
- this is used for **plagiarism detection!**
- see also: A. Ahtiainen, S. Surakka, and M. Rahikainen. 2006. Plaggie: GNU-licensed source code plagiarism detection engine for Java exercises. In *Proceedings* Baltic Sea '06. ACM, New York, NY, USA, 141-142. http://doi.acm.org/10.1145/1315803.1315831