

Quality Assurance Recapitulation

Part II

Test & Exam

- There will definitely be a question about
 - Drawing a class diagram
 - Drawing a screen diagram
 - Writing a JET generator
 - Using some simple Java reflection
- There will be general text questions about important concepts
- Prepare yourself by answering the quiz questions
- Each point in the test/exam roughly corresponds to one minute of estimated work
- Exam: half Ewan's part, half my part

**Don't
Panic**

Old Test

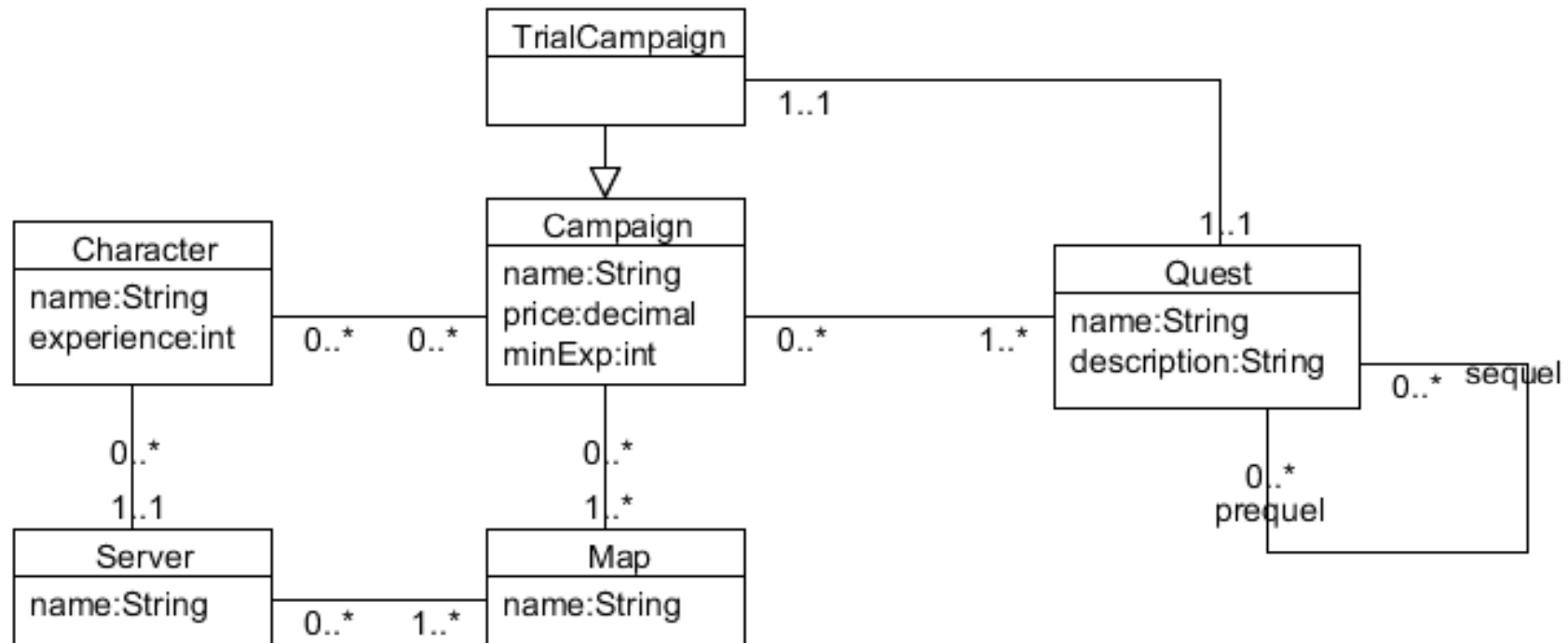


*"The past is behind, learn from it.
The future is ahead, prepare for it.
The present is here, live it."*

Question 1: Analysis Data Model

An online role playing game company wants you to design a system for managing game content. Characters in the game have a name and a number of experience points. Each character is located on exactly one game server, and the servers also have a name. Characters can participate in campaigns, which have a name and a minimum number of experience points that are required to play the campaign. Since normal campaigns have to be bought by gamers before they can be played, they also have a price. Quests are the parts of a campaign, and each campaign has at least one quest. A quest has a name and a description. Some quests are sequels of other quests, i.e. the other quests which are their prequels need to be completed first. Some campaigns are available as a trial campaign, which means that they can be played for free. A trial campaign offers only exactly one of the quests of the full campaign, to give gamers an impression before they buy the full campaign. Campaigns have at least one map. A map has a name and represents a location in the game. Each server has at least one of the maps.

Question 1: Analysis Data Model

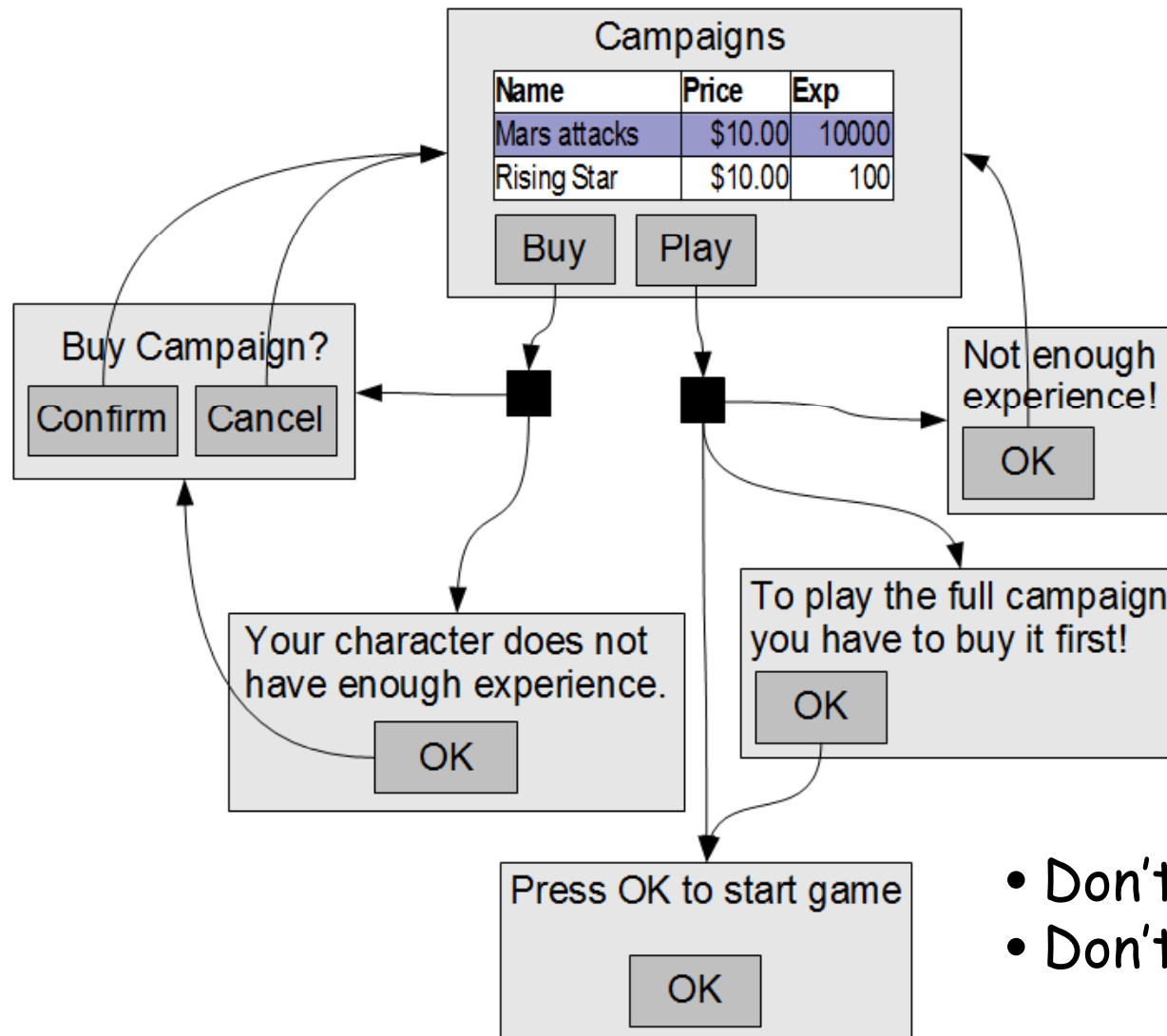


- **Associations:**
 - undirected (just lines), with min..max on both ends
- **Attributes with types (only those mentioned in the spec!)**
- **No unmentioned class (only those mentioned in the spec!)**
- **No association implementations; no methods**

Question 2: UI Model

The first screen shows a list of all available campaigns, with their attributes. Users can select a campaign, and choose to buy it or to play it. If they choose to buy it and their game character has enough experience for the campaign, they have to confirm their purchase in a confirmation dialogue, and are sent back to the list of campaigns. If their game character does not have enough experience, they are first shown a warning dialogue, and then they can confirm the purchase in the confirmation dialogue. If they select a campaign and choose to play it, one of three things may happen. If their game character does not have enough experience to play the selected campaign, then a notification appears and they are led back to the campaign list. If they have not bought the campaign yet, then a warning appears notifying them that they can only play the trial version of the campaign, and then they are led to the game screen. Otherwise they are led directly to the game screen. The game screen can just be a blank screen in your diagram.

Question 2: UI Model



- Don't forget fake data
- Don't forget branches

Question 3: JET Generator

- Write a JET template that receives an object of type `Class` as argument and generates a subclass of that class.
- Given the `Class` object of the following example class `Foo`:
`class Foo { public int a; public String b; }`
- The generator should generate class `FooWithReset`:

```
class FooWithReset extends Foo {  
    public Foo defaultValue;  
    public void reset() {  
        if (defaultValue == null)  
            throw new RuntimeException(  
                "No default value set");  
        this.a = defaultValue.a;  
        this.b = defaultValue.b;  
    }  
}
```


JET Syntax

- JET directive

```
<%@ jet package="hello" class="HelloTemplate" %>
```

More attributes for...

- Importing packages `imports="java.io.* java.util.*"`
- Changing JET tags `startTag="<@" endTag="@>"`

- Expressions `<%= argument+"xyz" %>`

- Scriptlets

```
<% int x=0; x++; %>
```

```
<% if (dayTime.isMorning()) { %> Good Morning
```

```
<% } else { %> Good Afternoon <% } %>
```

- Predefined variables:

- **StringBuffer** `stringBuffer`: generator output
`<% stringBuffer.append("Hello again!"); %>`
- **Object** `argument`: parameter of generate method

Question 3: JET Generator

```
<%@ jet imports="java.lang.reflect.*"
      class=" WithResetGenerator" %>
<%Class c = (Class)argument;%>

class <%=c.getSimpleName()%>WithReset
  extends <%=c.getName()%> {
  public <%=c.getSimpleName()%> defaultValue;
  public void reset() {
    if (defaultValue == null)
      throw new RuntimeException(
        "No default value set");
    <%for(Field f : c.getFields()) {%>
    this.<%=f.getName()%> =
      defaultValue.<%=f.getName()%>;
    <%}%>
  }
}
```

General Questions

Answer in no more than 3 sentences each.





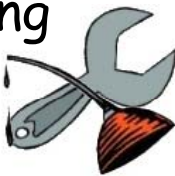
1. Give a brief description of CMM level 4.
2. What is reengineering?
3. What is the lost update problem in version control?
4. Explain how incorrect method overriding can cause problems.

Recap



*"There is no such thing
as tough. There is
trained and untrained.
Now which are you? "*

CMM

Level		Characteristics	Key process areas
1 Initial		Unstable environment Unpredictable, ad hoc	None
2 Managed		Management processes Based on experience	Requirements management, project planning, tracking and oversight, CM
3 Defined		Standardized, docu- mented process Effective SE practices	Process definition & focus, training program, SE, peer review
4 Quant. Managed		Measurement program Predictably high quality	Quantitative process management, software quality management
5 Optimizing		Process improvement Analyze defects Disseminate experience	Technology & process change management, defect prevention



Processes



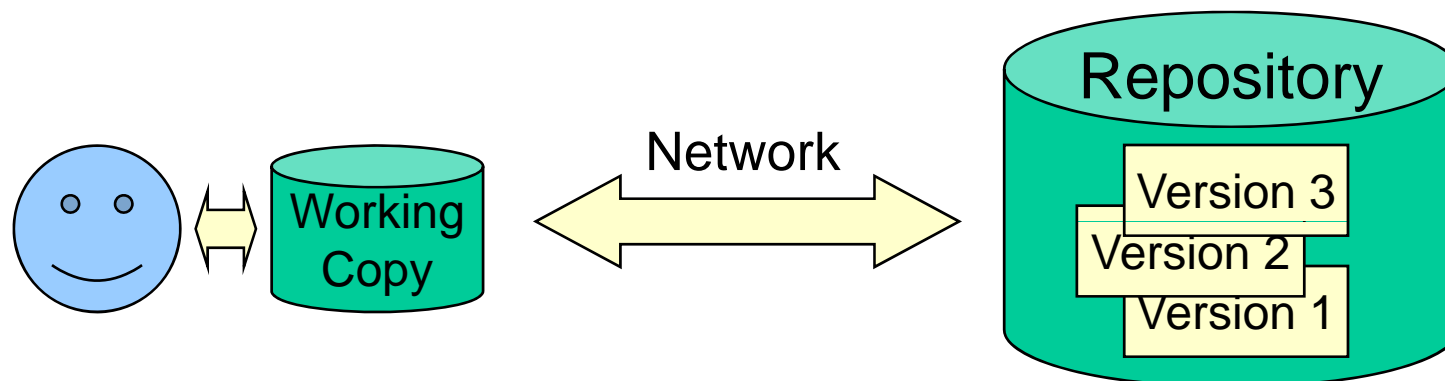
- **Adaptive vs. predictive Processes**
- **eXtreme Programming (XP)**
 - Agile process focused on programming as a team
 - **Short iterations**, as much feed back as possible
 - **12 best practices**: collective code ownership, refactoring, pair programming, ...
- **Rational Unified Process (RUP)**
 - **Heavyweight** process framework
 - Phases divided into iterations, several disciplines happening simultaneously
 - Best practices: risk & change management, use of tools, models & components

Modeling

- There are different levels of model data: **metamodels, models and model instances**
- **Domain Specific Languages (DSLs)** are languages for modeling particular domains
- Models are important for **forward and reverse engineering, re-engineering and round-trip engineering**
- **Meta-CASE** tools support the creation of graphical modeling tools
- **Class diagrams** specify the data model of a system
- **Screen diagrams** illustrate the UI of a system
- **Click dummies** can be used for early user feed-back

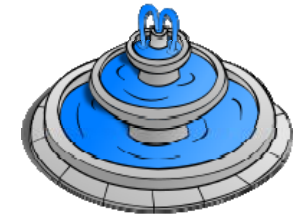
Version Control

- A Version Control System manages the different versions of all artefacts in a project
- Many **local working copies** and one **shared repository**, compressed with delta encoding
- Prevents **lost updates** through **locking** or **merging**
- Supports automatic merging and detects textual **conflicts**, but cannot detect non-textual semantic conflicts
- Conflicts always have to be **resolved** manually
- Subversion is a popular open-source VCS



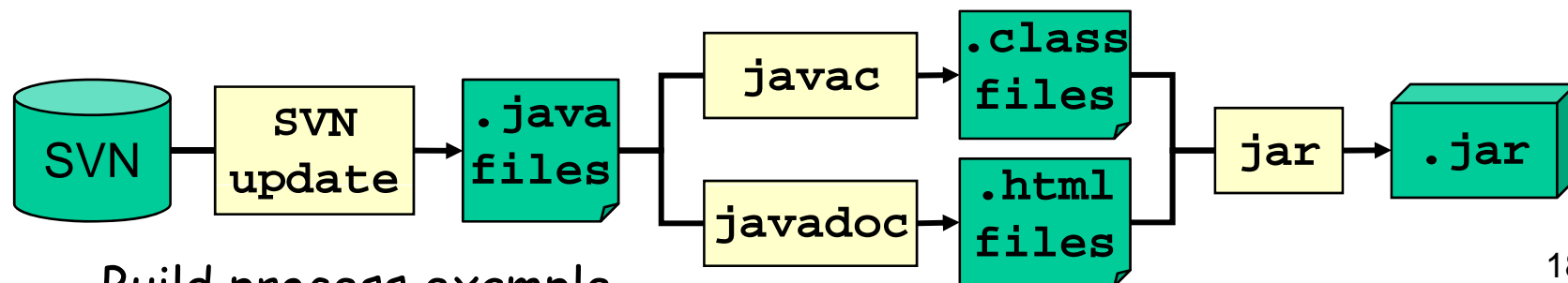
Version Control Best Practices

1. Complete **one change at a time** and commit it
 - If you committing several changes together you cannot undo/redo them individually
 - If you don't commit and your hard disk crashes...
2. **Don't break the build**
 - Test your changes before committing
3. Commit **only the source** files (e.g. not `.class` files)
4. **Use the log** by writing a summary for each commit
 - What has been changed and why
5. **Communicate** with the other developers
 - See who else is working on a part before changing it
 - Discuss and agree on a design
 - Follow the project guidelines & specifications



Build Tools

- **Build process:** generation of end-user artefacts from developer artefacts
- **Build tools** automate the build process
 - Manage different build configurations with **build scripts**
 - **Tasks** are pieces of code defining what is done
 - **Targets** define sets of tasks that belong together
 - Targets can **depend** on other targets
 - **Properties** can be used to configure a build script



Build process example

Reflection

- Reflection is the ability of a program to **observe** and possibly **modify** its own **structure** and **behavior**
 - Introspection and intercession
 - OO languages use **metaobject protocols (MOPs)** with **metaclasses** and **metaobjects**
- Java only supports some introspection and introspective access to methods and fields
- Other languages (e.g. MetaJ) offer full reflection
- However: reflection can be dangerous; it introduces new sources of error



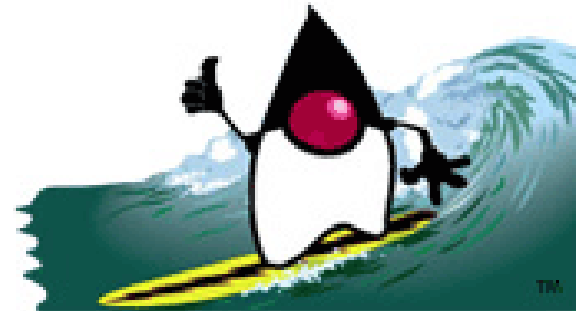
Generators



- **Generators** are programs that can generate certain artefacts
- **Generative programming** tries to generate program parts, e.g. class extensions, proxies, wrappers, interfaces
- With **templates** generator output can be given in its natural form
- **Generator type errors** are parts of the generator program that can potentially generate malformed code
- **Java Emitter Templates (JET)** are a popular generator technology for Eclipse

Coding Principles

- There are **common Java errors** one should be aware of, e.g.
 - Overriding mistakes (overriding with incorrect name or semantics)
 - Incomplete error handling
- There are **coding style** conventions for naming and other rules that should be followed
- **Refactoring** means to improve the design of existing code safely
 - Refactoring may change **HOW** the code works but **NOT WHAT** it does (preserving semantics)
 - **Simplicity** is one of the most important criteria for code



Good luck for the test!!!



*In the middle of difficulty
lies opportunity.
(Albert Einstein)*