# Software Engineering I (02161)

## Week 5

Assoc. Prof. Hubert Baumeister

DTU Compute
Technical University of Denmark

Spring 2018

# Contents

User Stories

Class Diagrams I

Version control

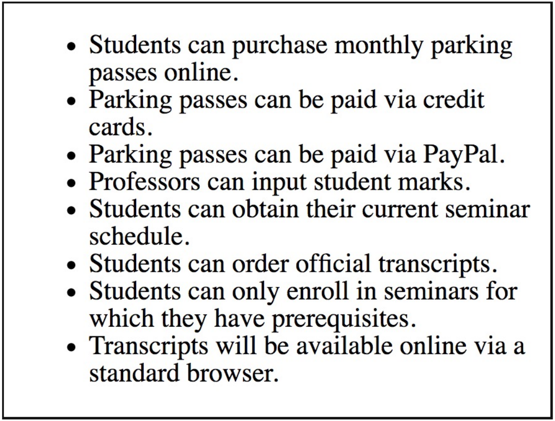
# User stories

- Requirements documentation for agile processes
  - Simplifies use cases
- Contains a ”story” that the user tells about the use of the system
- Focus on features
  - ”As a customer, I want to book and plan a single flight from Copenhagen to Paris”.
- functional + non-functional requirement

  e.g. ”The search for a flight from Copenhagen to Paris shall take less than 5 seconds”
- user story cards: index cards

# Example of user stories

Each line is one user story:



- Students can purchase monthly parking passes online.
- Parking passes can be paid via credit cards.
- Parking passes can be paid via PayPal.
- Professors can input student marks.
- Students can obtain their current seminar schedule.
- Students can order official transcripts.
- Students can only enroll in seminars for which they have prerequisites.
- Transcripts will be available online via a standard browser.

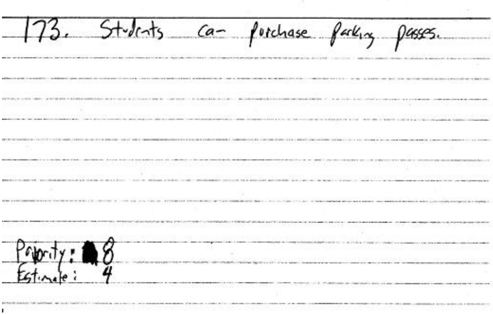
Scott Ambler 2003–2014 `http://www.agilemodeling.com/artifacts/userStory.htm`

# Example of user story cards

”Use the simplest tool possible”

→ index cards, post-its, …

- electronically: e.g. Trello (`trello.com`)

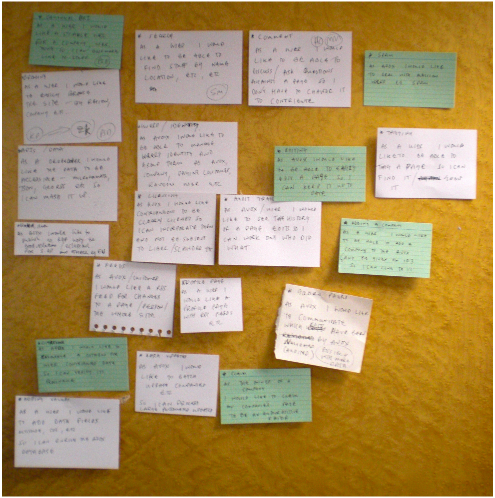


173. Students ca- purchase parking passes.

Priority: 8
Estimate: 4

Scott Ambler 2003–2014 `http://www.agilemodeling.com/artifacts/userStory.htm`

# Use the simplest tool possible



* CANONICAL URI
AS A USER I WOULD LIKE A STABLE URI FOR A COMPANY, USER, THING SO I CAN BOOKMARK LINK TO STUFF

* SEARCH
AS A USER I WOULD LIKE TO BE ABLE TO FIND STUFF BY NAME LOCATION, ETC, ETC

* COMMENT
AS A USER I WOULD LIKE TO BE ABLE TO DISCUSS / ASK QUESTIONS AGAINST A PAGE SO I DON'T HAVE TO CHANGE IT TO CONTRIBUTE

* SPAM
AS AVOX I WOULD LIKE TO DEAL WITH MALICIOUS USERS EG SPAM

BROWSING
AS A USER I WOULD LIKE TO EASILY BROWSE THE SITE — BY REGION, COMPANY ETC.

USERS / IDENTITY
AS AVOX I WOULD LIKE TO BE ABLE TO MANAGE USERS IDENTITY AND GROUP THEM AS AVOX, COMPANY, PAYING CUSTOMER, RANDOM USER ETC

* EDITING
AS AVOX I WOULD LIKE TO BE ABLE TO EASILY EDIT A PAGE SO I CAN KEEP IT UP TO DATE

* TAGGING
AS A USER I WOULD LIKE TO BE ABLE TO TAG A PAGE SO I CAN FIND IT / ~~[illegible]~~ GROUP IT

APIS / DATA
AS A DEVELOPER I WOULD LIKE THE DATA TO BE ACCESSIBLE — MICROFORMATS, JSON, GEORSS ETC SO I CAN MASH IT UP.

* LICENSING
AS AVOX I WOULD LIKE CONTRIBUTIONS TO BE CLEARLY LICENSED SO I CAN INCORPORATE THEM AND NOT BE SUBJECT TO LIBEL / SLANDER ETC

* AUDIT TRAIL
AS AVOX / USER I WOULD LIKE TO SEE THE HISTORY OF A PAGE EDITS SO I CAN WORK OUT WHO DID WHAT

[illegible]
As AVOX I would like to publish as RDF using the [illegible] / [illegible] for S AP and others eg EU

* ADDING A COMPANY
AS A USER I WOULD LIKE TO BE ABLE TO ADD A COMPANY TO THE AVOX (AND BE GIVEN AN ID) SO I CAN LINK TO IT

* FEEDS
AS AVOX / CUSTOMER I WOULD LIKE A RSS FEED FOR CHANGES TO A PAGE / PERSON / THE WHOLE SITE.

PROFILE PAGE
AS A USER I WOULD LIKE A PROFILE PAGE WITH RSS FEEDS ETC

* GOLDEN PAGES
AS AVOX I WOULD LIKE TO COMMUNICATE WHICH ~~[illegible]~~ EDIT HAVE BEEN ~~REVIEWED~~ BY AVOX VALIDATED (AUDITED) (POSSIBLY WITH META DATA)

* CITATION
AS AVOX I WOULD LIKE TO EVALUATE A CITATION FOR USER CONTRIBUTED DATA SO I CAN VERIFY ITS PROVENANCE

* BATCH UPDATES
AS AVOX I WOULD LIKE TO BATCH UPDATE COMPANIES ETC SO I CAN PROCESS LARGE AUTOMATED UPDATES

* CLAIM
AS THE OWNER OF A COMPANY I WOULD LIKE TO CLAIM MY COMPANIES PAGE TO BE AN AUTHORITATIVE EDITOR

* ADDING VALUES
AS A USER I WOULD LIKE TO ADD DATA FIELDS ADDRESS, CEO, ETC SO I CAN ENRICH THE AVOX DATABASE

Paul Downey 2009 `https://www.flickr.com/photos/psd/3731275681/in/photostream/`

# MoSCoW method for prioritizing requirements

**Must have**: Minimal usable subset to achieve the Minimal Vialble Product

**Should have**: Important requirments but not time critical, i.e. not relevant for the current delivery time frame

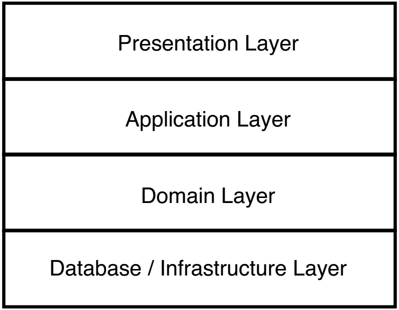
**Could have**: Desireable features; e.g. can improve usability

**Won't have/Would like**: Features explicitly excluded for the current delivery time frame

Wikipedia: `https://en.wikipedia.org/wiki/MoSCoW_method`

# Reminder: Two different ways of building the system

Build the system by layer/framework (traditional approach)



Presentation Layer

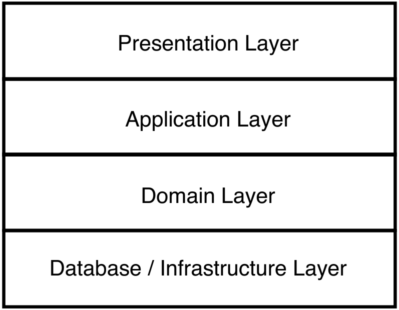
Application Layer

Domain Layer

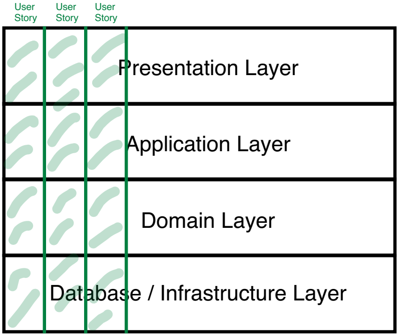
Database / Infrastructure Layer

# Reminder: Two different ways of building the system

Build the system by layer/framework (traditional approach)



Build the system by functionality (Agile approach)



→ User story driven: After every implemented user story a functional system

# Comparision: User Stories / Use Cases

User Case

- several abstract scenarios with one goal
- only functional requirements

Use Story

- one concrete scenario/feature
- Alternative scenarios of a use case are their own user story
- functional + non-functional requirement

  e.g. ”The search for a flight from Copenhagen to Paris shall take less than 5 seconds”

# Comparision: User Stories / Use Cases

Use Case

- Advantage
  - Overview over the functionality of the system
- Disadvantage
  - Not so easy to do a use case driven development
  - E.g. Login use case

Use Story

- Advantage
  - Easy software development process: user story driven
- Disadvantage
  - Overview over the functionality is lost

# Example: Login

Use case

name: Login

actor: User

main scenario

1 User logs in with username and password

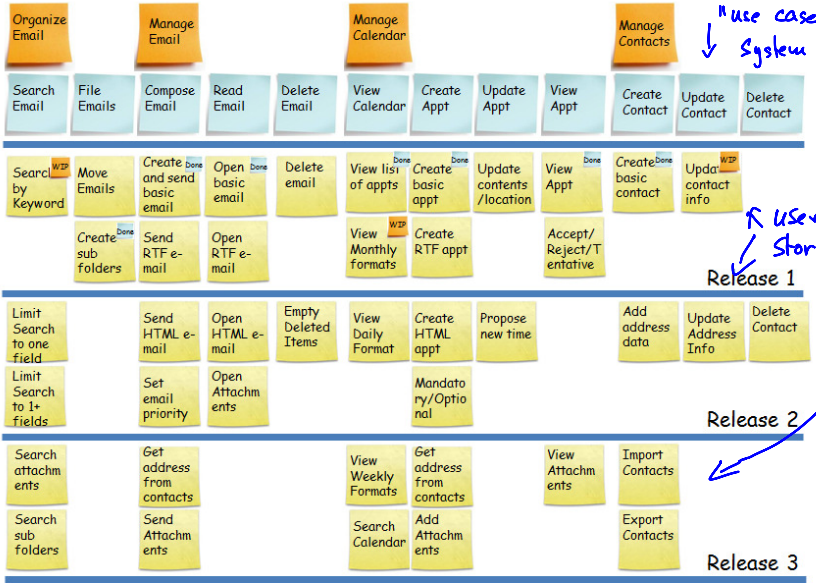
alternative scenario

1' User logs in with NEMID

User stories

1 User logs in with username and password
2 User logs in with NEMID

# User Story Maps



| Organize Email | | Manage Email | | | Manage Calendar | | | | Manage Contacts | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Search Email | File Emails | Compose Email | Read Email | Delete Email | View Calendar | Create Appt | Update Appt | View Appt | Create Contact | Update Contact | Delete Contact |
| Searcl WIP by Keyword | Move Emails | Create Done and send basic email | Open Done basic email | Delete email | View lisi Done of appts | Create Done basic appt | Update contents /location | View Done Appt | Create Done basic contact | Upda WIP contact info | |
| | Create Done sub folders | Send RTF e-mail | Open RTF e-mail | | View WIP Monthly formats | Create RTF appt | | Accept/Reject/Tentative | | | |
| | | | | | | | | | | | Release 1 |
| Limit Search to one field | | Send HTML e-mail | Open HTML e-mail | Empty Deleted Items | View Daily Format | Create HTML appt | Propose new time | | Add address data | Update Address Info | Delete Contact |
| Limit Search to 1+ fields | | Set email priority | Open Attachments | | | Mandatory/Optional | | | | | |
| | | | | | | | | | | | Release 2 |
| Search attachments | | Get address from contacts | | | View Weekly Formats | Get address from contacts | | View Attachments | Import Contacts | | |
| Search sub folders | | Send Attachments | | | Search Calendar | Add Attachments | | | Export Contacts | | |
| | | | | | | | | | | | Release 3 |

"use case" business level

"use case" system level

user stories

Shrikant Vashishtha `http://www.agilebuddha.com/wp-content/uploads/2013/02/IMAG0144.png`

# Combining Use Cases and User Stories

1. Use cases:
   - Gives an overview over the possible interactions

   → use case diagram
2. Derive user stories from use case scenarios (i.e. main- and alternative)
3. Implement the system driven by user stories
   - Note that different scenarios in use cases may have different priorities
     → Not necessary to implement all scenarios of a use case immediately

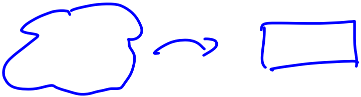
# Contents

User Stories

Class Diagrams I

Version control

# UML



- Unified Modelling Language (UML)
- Set of graphical notations: class diagrams, state machines, sequence diagrams, activity diagrams, . . .
- Developed in the 90’s
- ISO standard

# Class Diagram

- Structure diagram of object oriented systems
- Possible level of details

  Domain Modelling: typically low level of detail

  ⋮ Middle level

  Implementation: typically high level of detail
- Purpose:
  - Documenting the domain
  - Documenting the design of a system
  - A language to talk about designs with other programmers

# Why a graphical notation?

```
public class Assembly
           extends Component {
 public double cost() {   }
 public void add(Component c) {}
 private Collection<Component>
      components;
}

public class CatalogueEntry {
 private String name = "";
 public String getName() {}
 private long number;
 public long getNumber() {}
 private double cost;
 public double getCost() {}
}
```

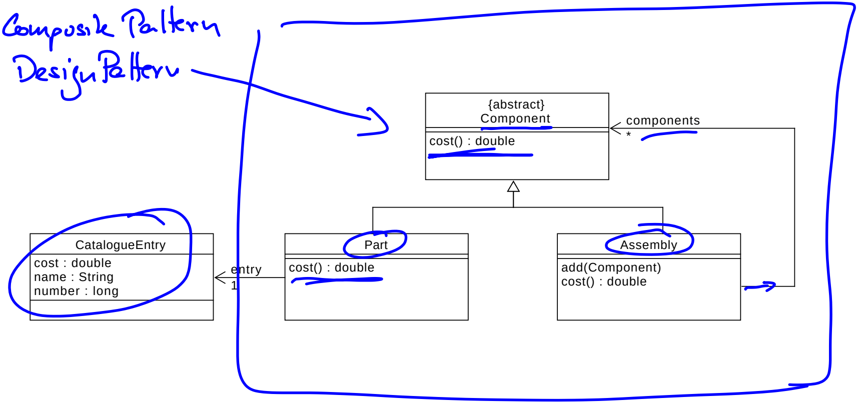
```
public abstract class Component {
 public abstract double cost();
}

public class Part extends Component
 private CatalogueEntry entry;
 public CatalogueEntry getEntry() {}
 public double cost(){}
 public Part(CatalogueEntry entry){}
```

# Why a graphical notation?

Composite Pattern

a Design Pattern



{abstract}
Component

cost() : double

components
*

Part

cost() : double

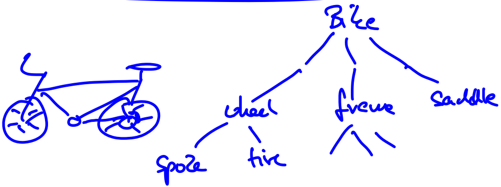
Assembly

add(Component)
cost() : double

CatalogueEntry

cost : double
name : String
number : long

entry
1



# General correspondence between Classes and Programs

«interface»

{abstract}
Medium

constraint

visibility

default +

italics

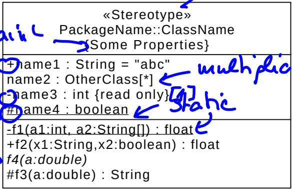
multiplicities

[1]

static

«interface»
List

size
add
contains
...



«Stereotype»
PackageName::ClassName
{Some Properties}

+name1 : String = "abc"
name2 : OtherClass[*]
-name3 : int {read only}
#name4 : boolean

-f1(a1:int, a2:String[]) : float
+f2(x1:String,x2:boolean) : float
*f4(a:double)*
#f3(a:double) : String

name2: ~~List<OtherClass>~~

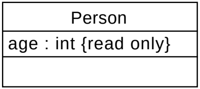
name2: OtherClass[*] ✓

name2: OtherClass[*]
{ordered}

```
package packagename
public class ClassName
{
   private String name1 = "abc";
   public List<OtherClass> name2 = new ArrayList<OtherClass>();
   private int name3;
   protected static boolean name4;

   private static float f1(int a1, String[] a2) { ... }
   public void f2(String x1, boolean x2) { ... }
   abstract public void f4(a:double);
   protected String f3(double a) { ... }
}
```

# Java: Public attributes



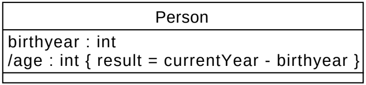
Person

age : int {read only}

```
public class Person {
   public int age;
}

for (Person p : persons) {
   System.out.println("age = ",p.age);
}
```

derived



Person

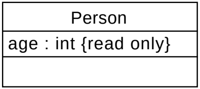
birthyear : int

/age : int { result = currentYear - birthyear }

```
public class Person {
   public int birthyear;
   public int age;
}

for (Person p : persons) {
   System.out.println("age = ",p.age);
}
```

# Java: Private attributes and getter and setter

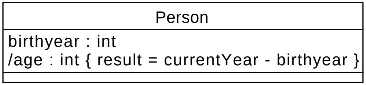


Person

age : int {read only}

```
public class Person {
   private int age;
   public int getAge() { return age; }
}

for (Person p : persons) {
   System.out.println("age = ",p.getAge());
}
```



Person

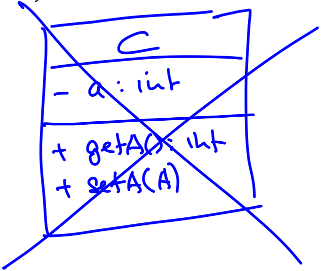
birthyear : int
/age : int { result = currentYear - birthyear }

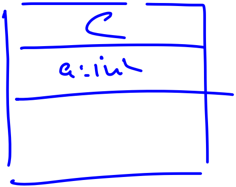
```
public class Person {
   private int birthyear;
   ~~private int age~~;
   public int getAge() { return ... ; }
}

for (Person p : persons) {
   System.out.println("age = ",p.getAge());
}
```

# Class Diagram and Program Code

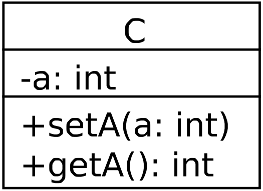
```
public class C {
  private int a;
  public int getA() { return a; }
  public void setA(int a) { this.a = a; }
}
```





# Class Diagram and Program Code

```
public class C {
  private int a;
  public int getA() { return a; }
  public void setA(int a) { this.a = a; }
}
```



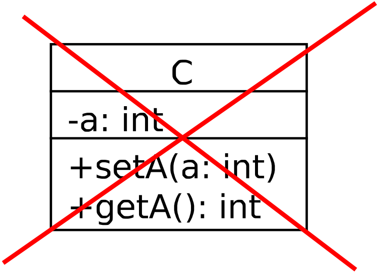
C

-a: int

+setA(a: int)
+getA(): int

# Class Diagram and Program Code

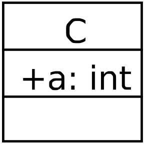
```
public class C {
  private int a;
  public int getA() { return a; }
  public void setA(int a) { this.a = a; }
}
```



~~C~~

~~-a: int~~

~~+setA(a: int)~~
~~+getA(): int~~



C

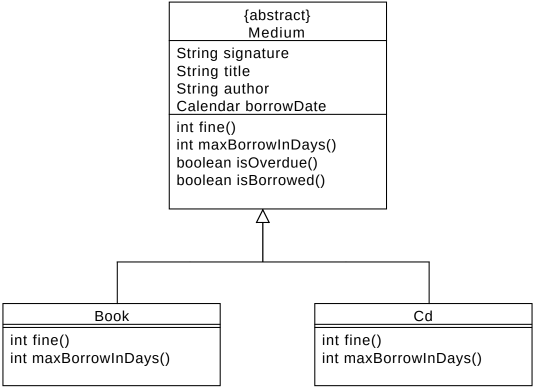
+a: int

# Generalization / Inheritance

- Programming languages like Java: Inheritance

```
abstract public class Medium { ... }
public class Book extends Medium { ... }
public class Cd extends Medium { ... }
```

- UML: Generalization / Specialization



{abstract}
Medium

String signature
String title
String author
Calendar borrowDate

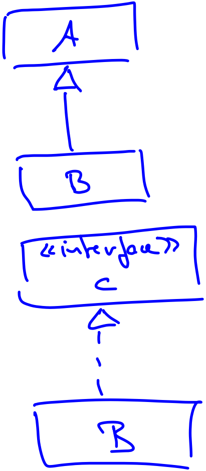
int fine()
int maxBorrowInDays()
boolean isOverdue()
boolean isBorrowed()

Book

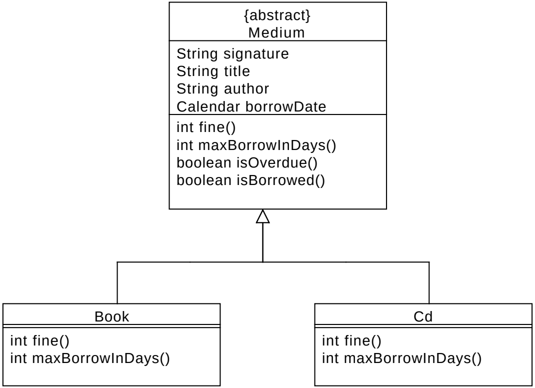
int fine()
int maxBorrowInDays()

Cd

int fine()
int maxBorrowInDays()



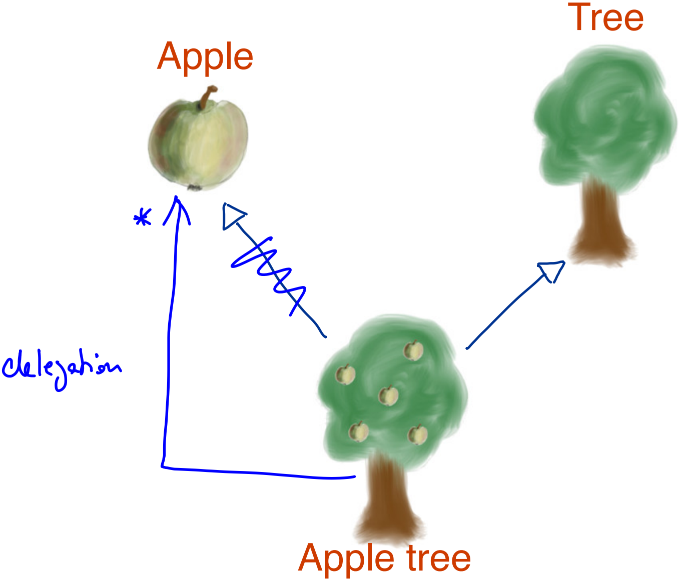
# Generalisation Example



Liskov-Wing Substitution Principle

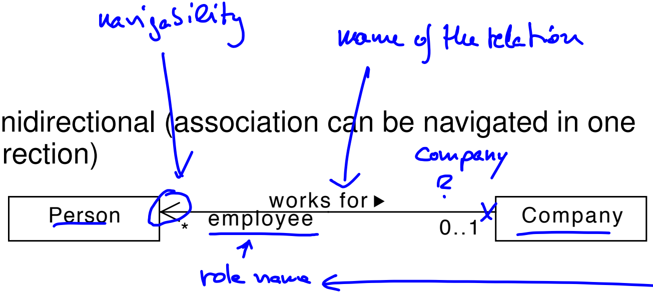
> ”*If S is a subtype of T, then objects of type T in a program may be replaced with objects of type S without altering any of the desirable properties of that program (e.g., correctness).*”

# Appletree



# Associations between classes

- Unidirectional (association can be navigated in one direction)



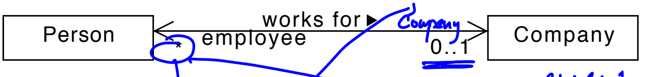
- Company has a field employees

```
public class Person
{
    ....
}
```

```
public class Company
{   ....
    private Set<Person> employees;
    ....
}
```

# Associations between classes

- Bidirectional (association can be navigated in both directions)

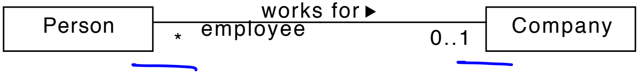


↳ multiplicity

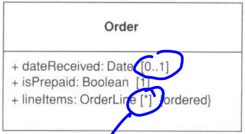
```
public class Person
{   ....
    private Company company;
    public getCompany() {
      return company;
    }
    public setCompany(Company c) {
      company = c;
    }
    ....
}
```

```
public class Company
{   ....
    private Set<Person> employees;
    ....
}
```

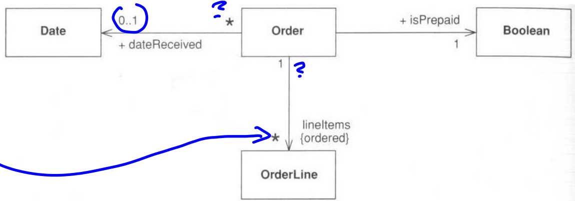
- Bidirectional or no explicit navigability
  - no explicit navigability ≡ no fields



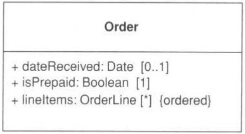
# Attributes and Associations


Order
+ dateReceived: Date [0..1]
+ isPrepaid: Boolean [1]
+ lineItems: OrderLine [*] {ordered}

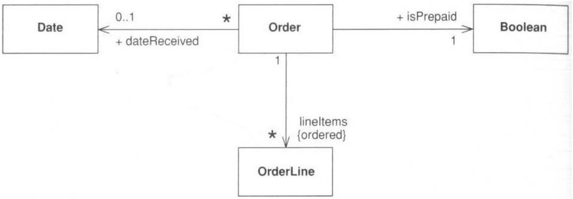
```
public class Order {
  private Date date;
  private boolean isPrepaid = false;
  private List<OrderLine> lineItems =
    new ArrayList<OrderLine)();
  ...
}
```

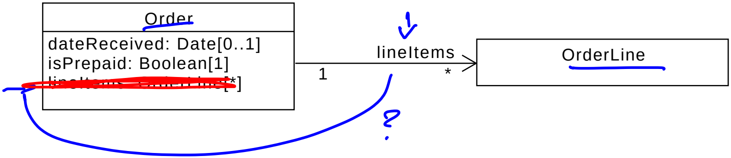


# Attributes and Associations


Order
+ dateReceived: Date [0..1]
+ isPrepaid: Boolean [1]
+ lineItems: OrderLine [*] {ordered}

```
public class Order {
  private Date date;
  private boolean isPrepaid = false;
  private List<OrderLine> lineItems =
    new ArrayList<OrderLine)();
  ...
}
```





# Contents

User Stories

Class Diagrams I

Version control

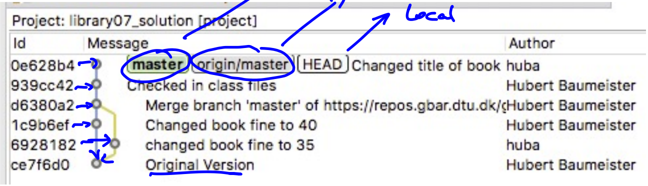
# What is version control?

Version Control

- Snapshots of project files (e.g. .java files)
- Project History
- Project Backup
- Concurrent work on project files
- Various systems: Git, Concurrent Versions System (CVS), Subversion (SVN), Team Foundation Server (TFS) ...

# Git

- Developed by Linus Torvalds for Linux
- Command line tools but also IDE support
- Commit: Snapshot of the project
- Commit: differences to previous snapshot + pointer to snapshot
- Names of commits: SHA1 hashes of their contents
  - 63d281344071f3ae1054bca63f1117f76a3d5751
  - short 63d2813



Project: library07_solution [project]

| Id | Message | Author |
|---|---|---|
| 0e628b4 | master origin/master HEAD Changed title of book | huba |
| 939cc42 | Checked in class files | Hubert Baumeister |
| d6380a2 | Merge branch 'master' of https://repos.gbar.dtu.dk/( | Hubert Baumeister |
| 1c9b6ef | Changed book fine to 40 | Hubert Baumeister |
| 6928182 | changed book fine to 35 | huba |
| ce7f6d0 | Original Version | Hubert Baumeister |

- Branch: Two commits with same parent
- Merging branches: Merging the changes of two commits into one

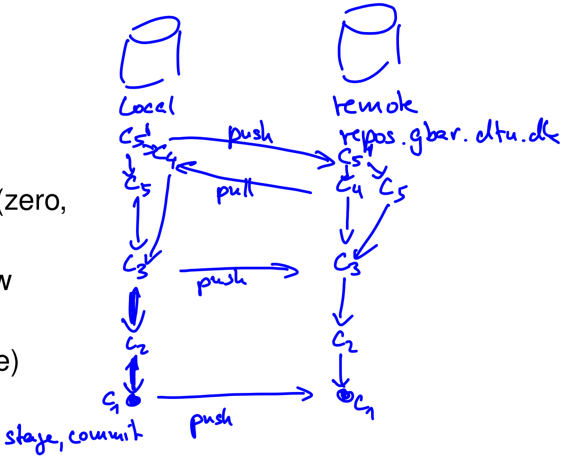
# Git: Distributed repository

- Local repository
- Remote repositories (zero, one or more)

→ **Stage** + **commit** (new local snapshot)

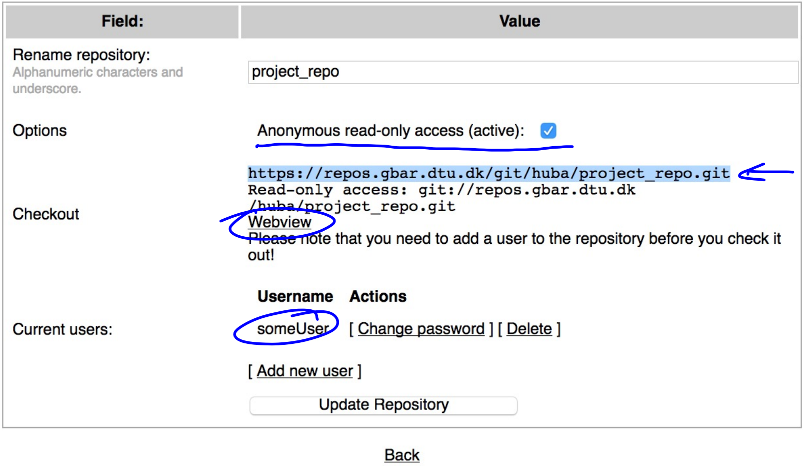
→ **Push** (local → remote)

→ **Pull** (remote → local)



# Starting with a project

1 Create a central repository: `http://repos.gbar.dtu.dk`



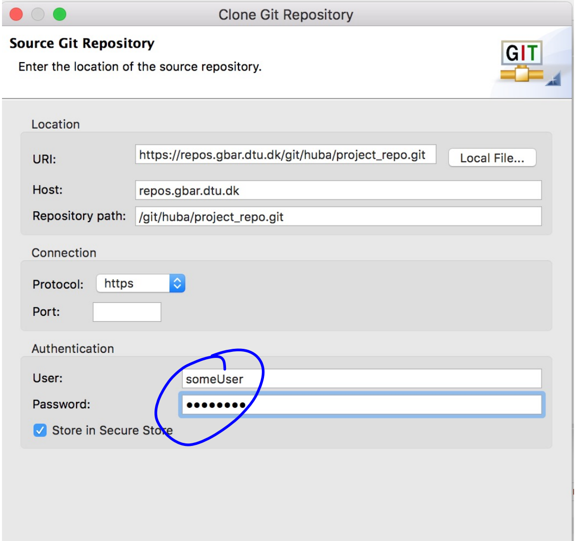
| Field: | Value |
|---|---|
| Rename repository:<br>Alphanumeric characters and underscore. | project_repo |
| Options | Anonymous read-only access (active): ☑ |
| Checkout | https://repos.gbar.dtu.dk/git/huba/project_repo.git<br>Read-only access: git://repos.gbar.dtu.dk /huba/project_repo.git<br>Webview<br>Please note that you need to add a user to the repository before you check it out! |
| Current users: | **Username** **Actions**<br>someUser [ Change password ] [ Delete ]<br>[ Add new user ] |
|  | Update Repository |

Back

# Starting with a project

2 Open Git perspective in Eclipse (Window::Perspective::Open Perspective::Other::Git)

3 Paste repository URL in ”Git Repositories” window



**Source Git Repository**

Enter the location of the source repository.

Location

URI: https://repos.gbar.dtu.dk/git/huba/project_repo.git

Host: repos.gbar.dtu.dk

Repository path: /git/huba/project_repo.git

Connection

Protocol: https

Port:

Authentication

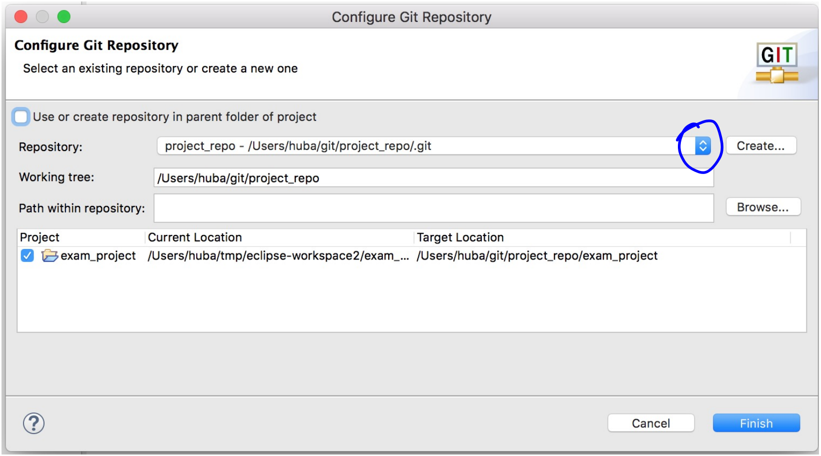
User: someUser

Password: ●●●●●●●●●

☑ Store in Secure Store

# Starting with a project

2. Create an initial project in Eclipse
3. Team::Share Project:



Configure Git Repository

**Configure Git Repository**

Select an existing repository or create a new one

□ Use or create repository in parent folder of project

Repository: project_repo - /Users/huba/git/project_repo/.git

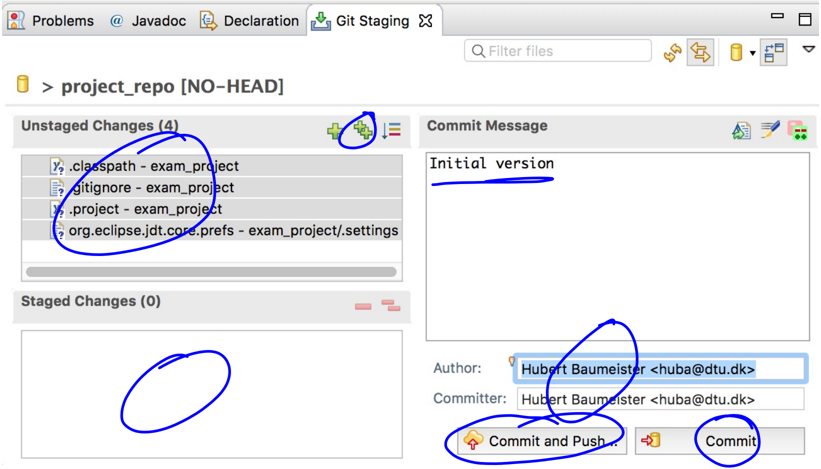
Working tree: /Users/huba/git/project_repo

Path within repository:

| Project | Current Location | Target Location |
|---|---|---|
| ☑ exam_project | /Users/huba/tmp/eclipse-workspace2/exam_... | /Users/huba/git/project_repo/exam_project |

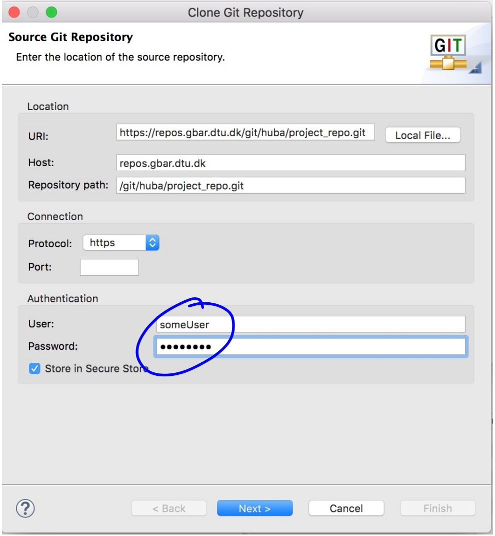
# Starting with a project

4 Stage changed files / commit (/ push)



# Starting with a project

5 Clone the repository from the central repository: Git repository view



Clone Git Repository

**Source Git Repository**

Enter the location of the source repository.

Location

URI: https://repos.gbar.dtu.dk/git/huba/project_repo.git

Host: repos.gbar.dtu.dk

Repository path: /git/huba/project_repo.git

Connection

Protocol: https

Port:

Authentication

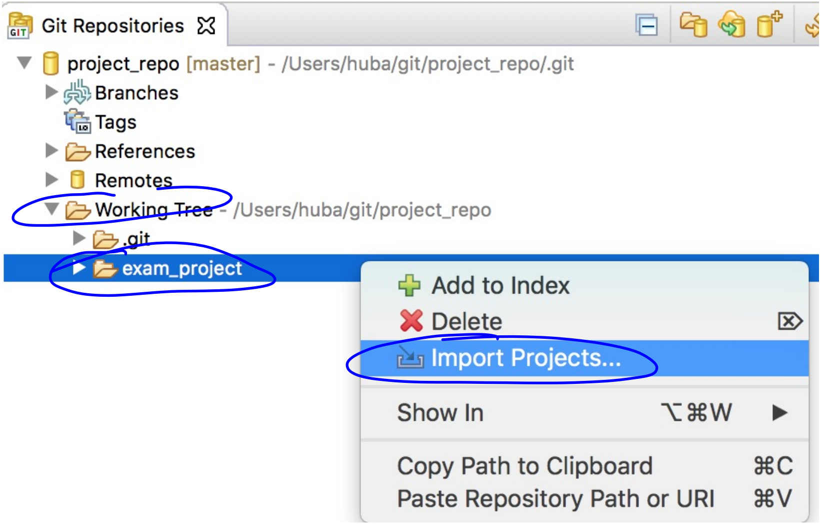
User: someUser

Password: ••••••••

☑ Store in Secure Store

# Starting with a project

## 6 Import projects



Git Repositories
project_repo [master] - /Users/huba/git/project_repo/.git
Branches
Tags
References
Remotes
Working Tree - /Users/huba/git/project_repo
.git
exam_project
Add to Index
Delete
Import Projects...
Show In ⌥⌘W ▶
Copy Path to Clipboard ⌘C
Paste Repository Path or URI ⌘V

# Working with Git: Centralized Workflow

# Working with Git: Centralized Workflow

1. Pull the latest changes from the central repository
2. Work on a user story with commits to the local repository as necessary (Team::Commit)
3. Once the user story is done (all tests are green) stage and commit the result
4. Before pushing your commits first pull all commits done in the meantime by others from the central repository
   → this will merge their commits with the local ones and create a new merged commit
5. Fix any merge conflicts until all tests are green again
6. push your final commit to the central repository

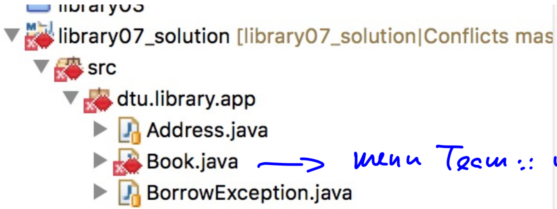
Important: Never push a commit where the tests are failing

Continous Integration: Merge often with the master branch

# When Pushing commits fail

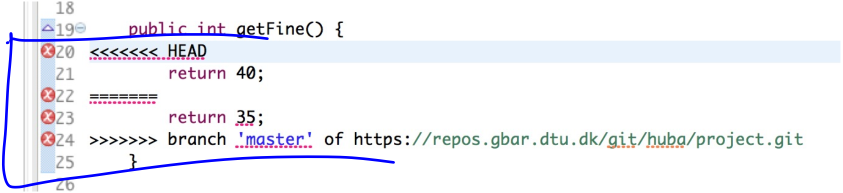
- Pushing fails if someone else as pushed his commits before: No fast-forward merge possible
    1. pull from central repository
        - this automatically tries to merge the changes,
    2. compile: fix possible compilation errors
    3. run the tests: fix failing tests
    4. commit and push again

# Merge conflicts when pulling



```
library03
▼ library07_solution [library07_solution|Conflicts mas
  ▼ src
    ▼ dtu.library.app
      ▶ Address.java
      ▶ Book.java
      ▶ BorrowException.java
```

→ menu Team :: merge tool



```
    public int getFine() {
<<<<<<< HEAD
        return 40;
=======
        return 35;
>>>>>>> branch 'master' of https://repos.gbar.dtu.dk/git/huba/project.git
    }
```

1. Resolve conflicts (option: Merge tool)
2. Stage your changes
3. Commit and push changes

# Working with Git: Feature Branch Workflow

# Working with Git: Feature Branch Workflow

- Create a branch for each feature, bug, group of work, etc.
- Only when the feature is done, merge to master branch
- Keeps master branch *clean*.
- Work on feature can be shared

# Git resources

- Git tutorial
  `https://www.sbf5.com/~cduan/technical/git/`
- Git Book: `https://git-scm.com/book/en/v2`

# Exam project

4

- Exam project
  - Week 06: Project introduction and forming of project groups (4); participation mandatory
  - Week 13: Demonstration of the projects (each project 10 min.) This is not an oral examination!
- Group forming
  - Group forming: **mandantory** participation in the lecture next week
    - Either you are **personally** present or someone can **speak for you**
    - *If not, then there is no guarantee for participation in the exam project*