HY 351: Ανάλυση και Σχεδίαση Πληροφοριακών Συστημάτων
CS 351: Information Systems Analysis and Design

UML: Introduction and Overview

How UML came up?
Overview of the UML Techniques and their uses
Why do analysis and design using UML ?
Hello World! in UML

Lecture : 3b
Date : 4-10-2005

Yannis Tzitzikas
University of Crete, Fall 2005

UML Introduction

- Successor to the wave of object-oriented analysis and design (OOA&D) methods that appeared in the late '80s and early '90s.
- Unifies the methods of
  - Booch
  - Rumbaugh (OMT)
  - Jacobson
- Now it is an OMG (Object Management Group) standard

How we got here?

- 1980: C++
  - Need to adapt the design methods of ('70s-'80s) for the object-oriented world
- 1989-91 "Recursive Design Approach" (Sally Shlaew, Steve Meller)
- P. Coad and Ed. Yourdon (books 1991, 1991b, 1995,1999)
- Responsibility-Driven Design (Wirfs-Brock 90)
- Class-Responsibility-Collaboration (CRC Cards) Beck and Cunnigham
- Grady Booch: work with Rational Software (for Ada systems)
- Jim Rumbaugh: Object-Modeling Technique (OMT)
- The most conceptual of these books: Martin and Odell, 94
- Ivar Jacobson (introduced the concept of use cases)
- Δεν υπήρχε διάθεση για τυποποίηση (standardization)
  - Κάθε ένας χρησιμοποιούσε τους δικούς του συμβολισμούς και μεθοδολογία

Famous joke:
- What is the difference between a methodologist and a terrorist?
- You can negotiate with a terrorist!

The birth of UML

- Jim Rumbaugh and G. Booch => Rational Software
- 1996: The 3 amigos (James Rumbaugh, Grady Booch, Ivar Jacobson)
  - **UML Version 1.1 Became OMG standard**
- Current version: UML Version 2.0, 2003

The objective of UML

To provide a common vocabulary of object-oriented terms and diagramming techniques that is rich enough to model any systems development project from analysis through implementation

How many UML diagrams exist?

- UML 2.0 defines 14 diagrammatic techniques used to model a system.
- Diagrams for modeling the structure of a system
  - Class, Object, Package, Deployment, Component, Composite Structure
- Diagrams for modeling the behavior of a system
  - Activity, Sequence, Communication, Interaction Overview, Timing, State, Protocol State Machine, Use Case Diagrams

When we use what diagram?

- Different diagrams are appropriate for different phases of the project
- Some diagrams can be used in more than one phase. They start from a very very abstract (and conceptual) form and evolve to include details that can even lead to code generation.

Notations and Meta-Models (αυστηρότητα έναντι ευχρηστίας)

- ULM: defines a notation and a meta-model
  - Notation: graphical stuff we see in models, i.e. syntax
- Question: What exactly is meant by each one symbol ?
  - i.e. what is a **class**, what is a **multiplicity** ?
- There is not a formal interpretation.
- Formal interpretations can be found in the area of formal methods
  - where design and specifications are represented using derivatives of predicate calculus

Why not natural language?

Too imprecise and gets tangled when comes to more complex concepts

Why not formal methods?

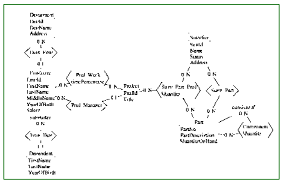
Even if we can prove that a program satisfies a mathematical specification, there is no way to prove that the mathematical specification actually meets the real requirements of the system.

Other problems of formal methods:

- Often lead to getting bogged down (βαλτώνω) in lots of minor details
- Hard to understand and manipulate
  - often harder to deal with that programming languages
  - **and you can't even execute them!**

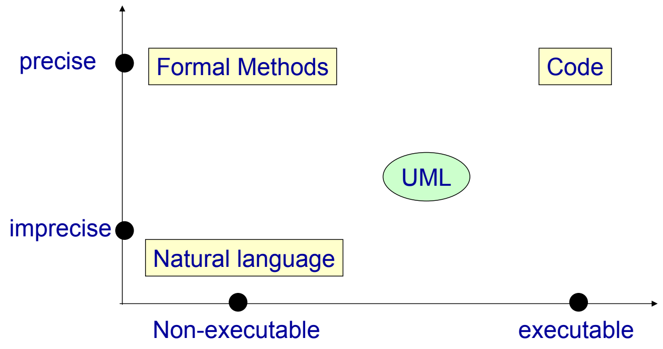
Why diagrams ?

```
create table Component (
    C_P_ID_Par char(10) not null,
    ID_Par char(10) not null,
    Quantity char(1) not null,
    constraint ID_Component primary key
        (C_P_ID_Par, ID_Par));
create table Deparment (
    ID_Dep char(10) not null,
    DepId char(1) not null,
    DepName char(1) not null,
    Address char(1) not null,
    constraint ID primary key (ID_Dep));
create table Dependent (
    FirstName char(1) not null,
    LastName char(1) not null,
    YearOfBirth char(1) not null,
    Supporter char(10) not null);
create table Employee (
    ID_Emp char(10) not null,
    EmpId char(1) not null,
    FirstName char(1) not null,
    LastName char(1) not null,
    MiddleName char(1) not null,
    YearOfBirth char(1) not null,
    Salary char(1) not null,
    ID_Dep char(10),
    constraint ID primary key (ID_Emp));
create table Part (
    ID_Par char(10) not null,
    PartNo char(1) not null,
    PartDescription char(1) not null,
    QuantityOnHand char(1) not null,
    constraint ID primary key (ID_Par));

create table Project (
    ID_Pro char(10) not null,
    ProjId char(1) not null,
    Title char(1) not null,
    ID_Emp char(10),
    constraint ID primary key (ID_Pro));
create table Proj_Work (
    ID_Emp char(10) not null,
    ID_Pro char(10) not null,
    timePercentage char(1) not null,
    constraint ID_Proj_Work primary key
        (ID_Pro, ID_Emp));
create table Supplier (
    ID_Sup char(10) not null,
    SupId char(1) not null,
    Name char(1) not null,
    Status char(1) not null,
    Address char(1) not null,
    constraint ID primary key (ID_Sup));
create table Supp_Part (
    ID_Par char(10) not null,
    ID_Sup char(10) not null,
    constraint ID_Supp_Part primary key
        (ID_Par, ID_Sup));
create table Supp_Part_Proj (
    ID_Par char(10) not null,
    ID_Pro char(10) not null,
    ID_Sup char(10) not null,
    Quantity char(1) not null,
    constraint ID_Supp_Part_Proj primary key
        (ID_Par, ID_Sup, ID_Pro));

-- Constraints Section
-- ____________________

alter table Component add constraint FKconsistsOf
    foreign key (ID_Par)
    references Part;
alter table Component add constraint FKCom_Par
    foreign key (C_P_ID_Par)
    references Part;
alter table Dependent add constraint FKEmp_Dep
    foreign key (Supporter)
    references Employee;
alter table Employee add constraint FKDept_Emp
    foreign key (ID_Dep)
    references Deparment;
alter table Project add constraint FKProj_Manager
    foreign key (ID_Emp)
    references Employee;
alter table Proj_Work add constraint FKPro_Pro
    foreign key (ID_Pro)
    references Project;
alter table Proj_Work add constraint FKPro_Emp
    foreign key (ID_Emp)
    references Employee;
alter table Supp_Part add constraint FKSup_Sup_1
    foreign key (ID_Sup)
    references Supplier;
alter table Supp_Part add constraint FKSup_Par_1
    foreign key (ID_Par)
    references Part;
alter table Supp_Part_Proj add constraint FKSup_Sup
    foreign key (ID_Sup)
    references Supplier;
alter table Supp_Part_Proj add constraint FKSup_Pro
    foreign key (ID_Pro)
    references Project;
alter table Supp_Part_Proj add constraint FKSup_Par
    foreign key (ID_Par)
    references Part;
```

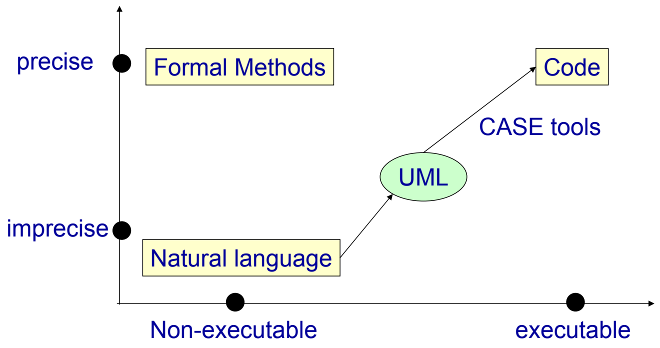


A picture is worth a thousands of words

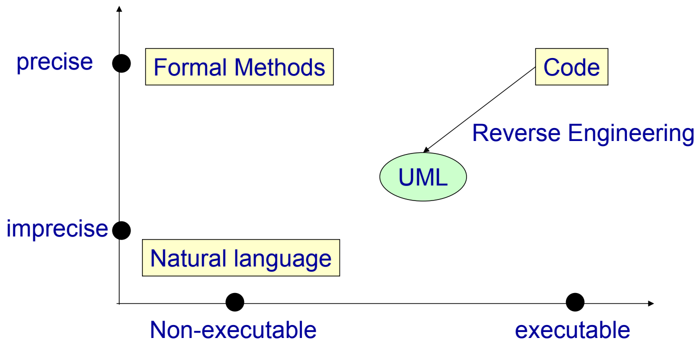
Overall picture



Overall picture

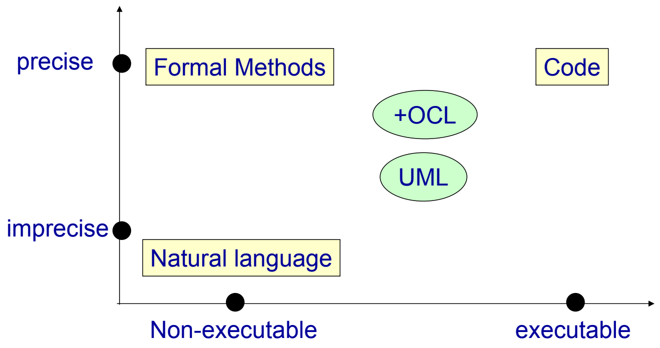


Overall picture



E.g. CodeLogic

Overall picture



Most OO methods have very little rigor

- Their notation appeals to intuition rather than formal definition
- This does not seem to have done much harm. These methods may be informal, but many people still find them useful - and it is usefulness that counts.
- However, OO people are looking for ways to improve the rigor of methods without sacrificing their usefulness
  - one way: to define a meta-model: a diagram, usually a class diagram, that defines the notation

How strictly should you stick to the modeling language?

- Depends on the purpose
  - in case you use a CASE tool that generates code, you have to stick to the CASE tool’s interpretation of the modeling language in order to get acceptable code
  - in case you use the diagrams for communication purposes, you have a little more leeway

List of UML Diagrammatic techniques and their uses

- **Use Case Diagram** (διάγραμμα περιπτώσεων χρήσης)
- **Class Diagram** (διάγραμμα κλάσεων)
- **Interaction Diagram** (διάγραμμα αλληλεπίδρασης)
  - Sequence Diagrams (διαγράμματα ακολουθίας)
  - Collaboration Diagrams (διαγράμματα συνεργασίας)
- **State Diagram** (διάγραμμα καταστάσεων)
- **Activity Diagram** (διαγράμματα δραστηριοτήτων)
- **Deployment Diagram** (διαγράμματα ανάπτυξης)
- **Package Diagram** (διάγραμμα πακέτων)
- **Component Diagram** (διάγραμμα εξαρτημάτων)

Use Cases

- **Use Case** = a set of scenarios tied together by a common user goal
- **Scenario** = a sequence of steps describing an interaction (user vs system)

Buy a Product

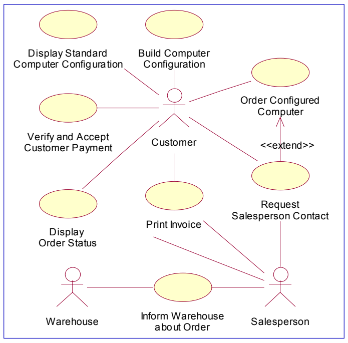
1. Customer browses through catalog and selects items to buy
2. Customer goes to check out
3. Customer fills in shipping information (address, next-day or 3-day delivery)
4. System presents full pricing information, including shipping
5. Customer fills in credit card information
6. System authorizes purchase
7. System confirms sale immediately
8. System sends confirming email to customer

Alternative: *Authorization Failure*
At step 6, system fails to authorize credit purchase
Allow customer to re-enter credit card information and re-try

UML Techniques
Use Case Diagrams (διαγρ. περιπτώσεων χρήσης)

Elicits requirements from users in meaningful chunks. Construction planning is build around delivering some use cases in each iteration. Basis for system testing.

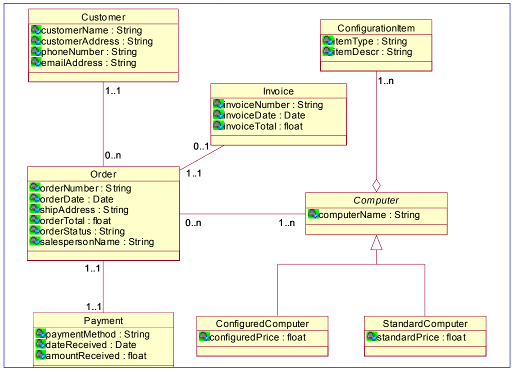
Used for: **Analysis**
Concerns: **Behavior**


Display Standard Computer Configuration
Build Computer Configuration
Order Configured Computer
Verify and Accept Customer Payment
Customer
<<extend>>
Request Salesperson Contact
Print Invoice
Display Order Status
Warehouse
Inform Warehouse about Order
Salesperson

UML Techniques
Class Diagrams (διαγράμματα κλάσεων)

Used for: **Analysis/Design**
Concerns: **Structure**

Shows static structure of concepts, types, and classes. *Concepts* show how users think about the world; *types* show interfaces of software components; *classes* show implementation of software components

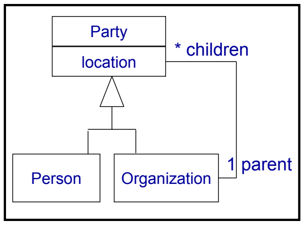


UML Techniques
Object Diagrams (διαγράμματα αντικειμένων)

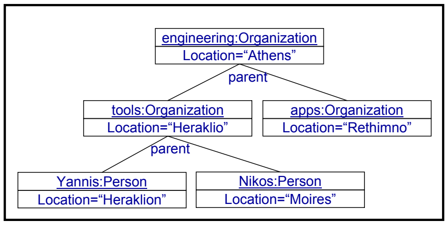
Shows the relationships between the objects in the system.

Used for: **Analysis/Design**
Concerns: **Structure**

Class diagram



Object diagram

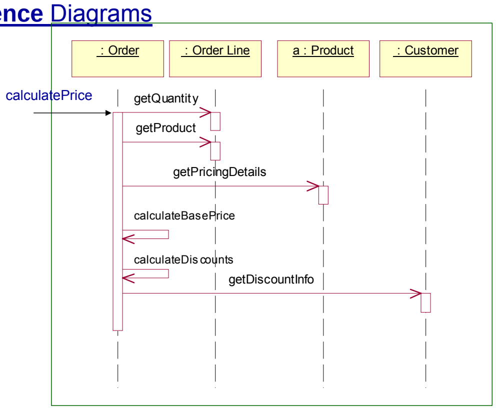


UML Techniques
Interaction Diagrams (διαγρ. αλληλεπίδρασης)

Used for: **Analysis/Design**
Concerns: **Behavior**

Show how several objects collaborate in a single use case

(A) **Sequence** Diagrams

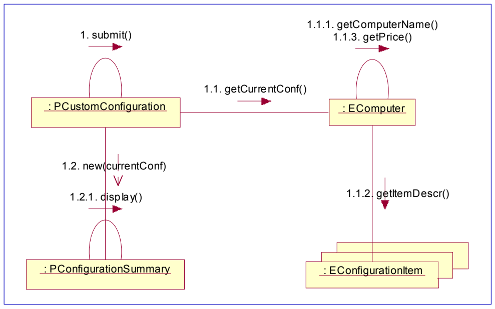


UML Techniques
Interaction

Used for: **Analysis/Design**
Concerns: **Behavior**

Show how several objects collaborate in a single use case

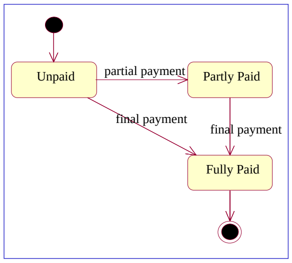
(B) **Collaboration** Diagrams

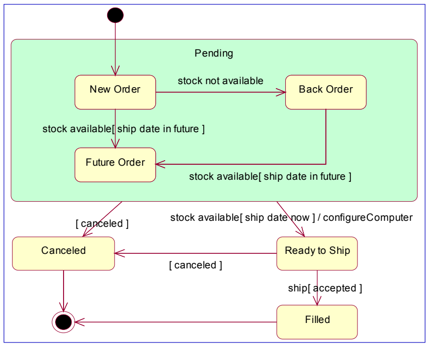


UML Techniques
State Diagrams (διάγρ. καταστάσεων)

Used for: **Analysis/Design**
Concerns: **Behavior**

Shows how <u>single object</u> behaves across <u>many use cases</u>



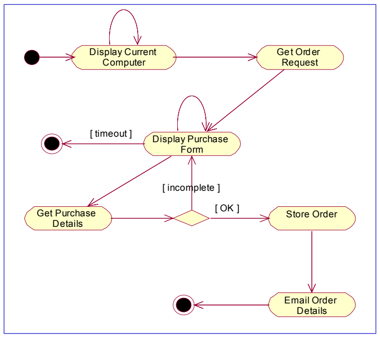


Pending
New Order
stock not available
Back Order
stock available[ship date in future]
Future Order
stock available[ship date in future]
[canceled]
stock available[ship date now] / configureComputer
Canceled
[canceled]
Ready to Ship
ship[accepted]
Filled

UML Techniques
Activity Diagrams (διαγρ. δραστηριοτήτων)

Used for: **Analysis/Design**
Concerns: **Behavior**

Illustrate business workflows independent of classes, the flow of activities in a use case, or detailed design of a method.

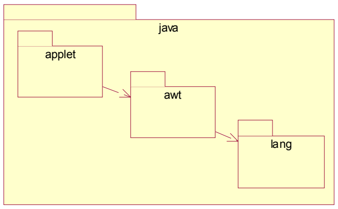


UML Techniques
Package Diagrams (διαγρ. πακέτων)

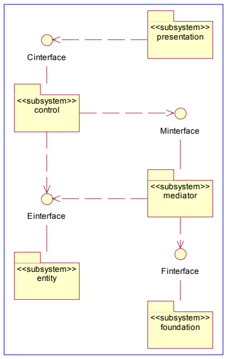
Used for: **Analysis/Design/Implement**
Concerns: **Structure**

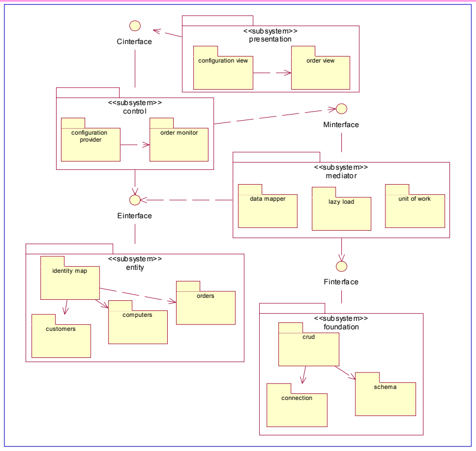
Shows <u>groups of classes</u> and <u>dependencies</u> among them

It can also group other UML elements together



UML Techniques
Package Diagrams

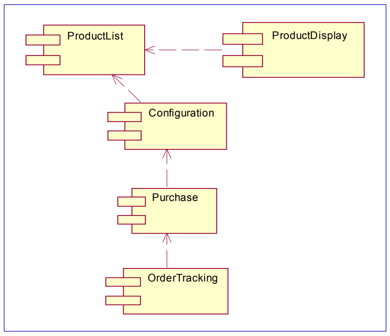




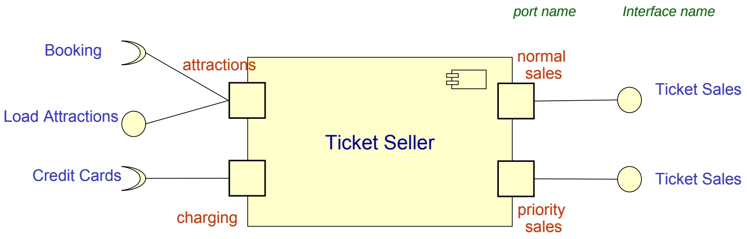
UML Techniques
Component Diagrams (διαγρ. εξαρτημάτω

Used for: **Phys./Design/Implement**
Concerns: **Structure**

- Component: a logical and <u>replaceable</u> part of a system that conforms to and provides the realization of a set of <u>interfaces.</u>



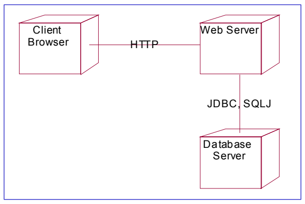
UML Techniques
Component Diagrams (διαγρ. εξαρτημάτων)



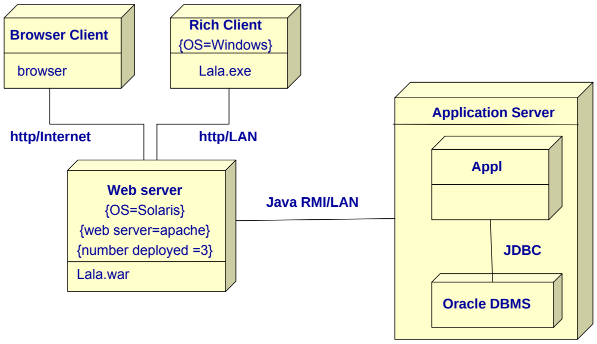
UML Techniques
Deployment Diagrams (διαγρ. ανάπτυξης)

Used for: **Phys./Design/Implement**
Concerns: **Structure**

Shows physical layout of components on hardware nodes



UML Techniques
Deployment Diagrams (διαγρ. ανάπτυξης)



Why do Analysis and Design using UML ?

- The real point of software development is executable code
  - diagrams are, after all, just pretty pictures
  - no user is going to thank you for pretty pictures; what a user wants is software that executes
- So we must ask ourselves
  - why we are using UML?
  - How it will help us when it comes down to writing the code ?
- Three main reasons
  - [A] Communication
  - [B] Learning OO
  - [C] Communication with Domain Experts

Why do Analysis and Design using UML ?
[A] Communication

- Fundamental reason to use UML
  - Natural language
    - too imprecise and gets tangled when comes to more complex concepts
  - Code
    - precise but too detailed
- So we use UML when we want a certain amount of precision but don't want to get lost in details
  - this doesn't mean avoid details, but use UML to highlight important details.

Why do Analysis and Design using UML ?
[A] Communication (II)

- Examples
  - You are a consultant to you want in a very short time to understand a big project
    - UML gives you an overall view of the system
    - class diagrams tell you what kinds of abstractions are used and where are the questionable parts (that need further work)
    - if you want a deeper view and see how classes collaborate, then you can see the interaction diagrams
  - You work for an organization as a system analyst/designer. You express your analysis and design using UML and then another company undertake the implementation.
- For the same reasons it is useful for the members of a project team
  - members have a common view (axon of reference)
  - new members enter the game quickly
  - less risk for the team if a person leaves the project

Why do Analysis and Design using UML ?
[B] Learning OO

- It takes time to learn and use well OO
  - CRC cards is a very useful technique to learn OO (not part of UML)
  - Interaction diagrams
    - make the message structure explicit and thus are useful for highlighting over-centralized designs
  - Class diagrams
    - quite similar to data models
    - danger: develop a class model that is data oriented rather than being responsibility oriented
  - Patterns:
    - gets you concentrate on good OO designs and to learn by following an example

Why do Analysis and Design using UML ?
[C] Communication with Domain Experts

- Use Cases:
  - US: a snapshot of one aspect of your system
  - The sum of all Use Cases: the external picture of your system
  - Very good tool to understand what users want
- Class diagrams
  - Help, especially those built from a “conceptual perspective”
- Activity diagrams
  - Useful if workflow processes are an important part of the user’s world
    - as they support parallel processes, can help you get away from unnecessary sequences

Hello World! in UML

mypage.html

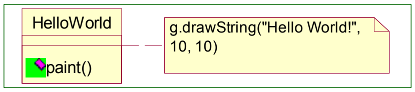
```
<html>
<body>
<APPLET CODE="HelloWorld.class">
</APPLET>
</body>
</html>
```

HelloWorld.java

```
import java.applet.Applet;
import java.awt.Graphics;

public class HelloWorld extends Applet {
   public void paint(Graphics g) {
      g.drawString("Hello world!", 50, 25);
   }
}
```

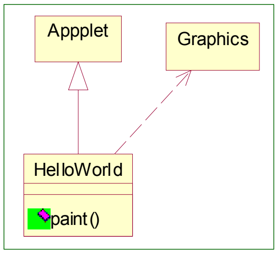
Hello World! in UML

```
import java.applet.Applet;
import java.awt.Graphics;

public class HelloWorld extends Applet {
   public void paint(Graphics g) {
      g.drawString("Hello world!", 50, 25);
   }
}
```



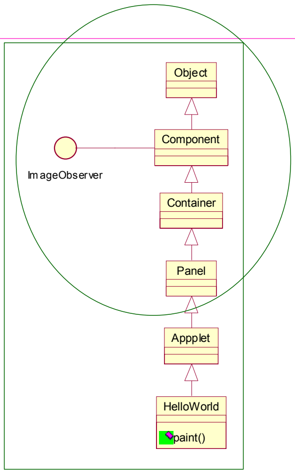
Hello World! in UML

```
import java.applet.Applet;
import java.awt.Graphics;

public class HelloWorld extends Applet {
   public void paint(Graphics g) {
      g.drawString("Hello world!", 50, 25);
   }
}
```



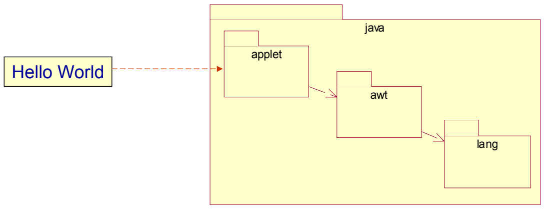
Hello World! in UML

By studying the Java libraries for Applet and Graphics the **entire hierarchy** is revealed:



Hello World! in UML

By studying how the Java libraries are **organized (packaged):**

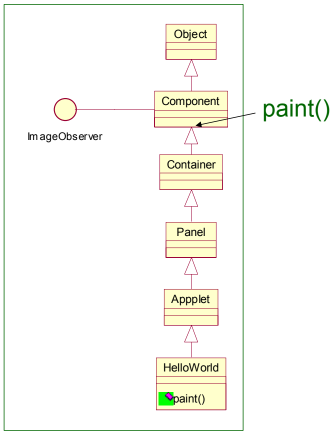


Hello World! in UML

How Java classes work together?

How the paint operation gets invoked ?

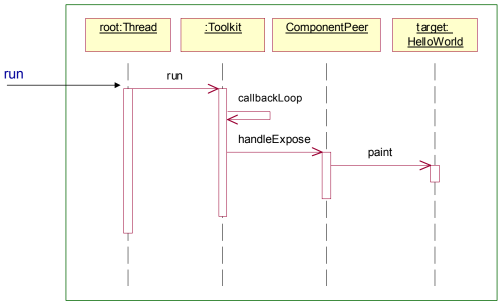
By studying Java libraries we see that paint is inherited from component


Object
Component
paint()
ImageObserver
Container
Panel
Appplet
HelloWorld
paint()

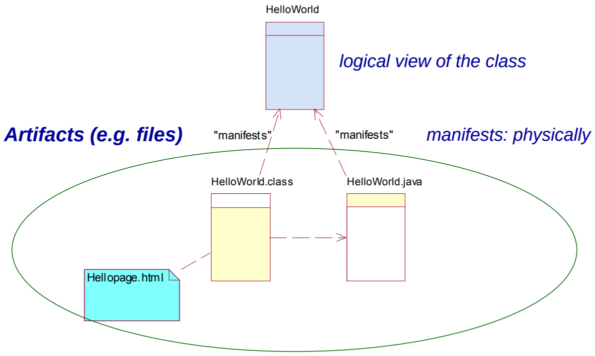
Hello World! in UML

By studying how Java classes work together we see that paint is invoked as follows:

paint is called as part of the thread that encloses applet.


root:Thread
:Toolkit
ComponentPeer
target: HelloWorld
run
run
callbackLoop
handleExpose
paint

Hello World! in UML: Physical view


HelloWorld
"manifests"
"manifests"
HelloWorld.class
HelloWorld.java
Hellopage.html

logical view of the class

Artifacts (e.g. files)

manifests: physically implements

Reading and References

- **UML Distilled: A Brief Guide to the Standard Object Modeling Language** (3rd Edition) by Martin Fowler, Addison Wesley, 2004.
- **The Unified Modeling Language User Guide** (2nd edition) by G. Booch, J. Rumbaugh, I. Jacobson, Addison Wesley, 2004