SE 450 Object-Oriented Software Development

Introduction

DePaul University

Outline

- Introduction
- Syllabus
- Introduction to Object Oriented Software Development

Introduction

- Me
- You

Introduction

- Dr. Wael Kessentini
  - PhD, University of Montreal, Canada
  - Main research interests
    - Software engineering
    - Software evolution
    - Software testing
    - Software quality
    - Software migration
    - Model-Driven engineering
    - …

Office Location: CDM 841
Office Phone: (312) 362-7080
Email: wkessent@depaul.edu
Office Hours: Tuesday 1:30 – 3:00pm

Introduction

- Me
- You

- Introduce yourself
  - Your background / experiences
  - What is your course load this semester?
  - Future plan (Dream job)

Instructor

- **Instructor: Dr. Wael Kessentini**
  - Office Location: CDM 841
  - Office Phone: (312) 362-7080
- Email: wkessent@depaul.edu
- Office Hours:
  - Tuesday 12:00 – 1:30pm

DePaul University

Outline

- Introduction
- Syllabus
- Introduction to Object Oriented Software Development

DePaul University

Topic of Interests

- Object-oriented design techniques
- UML diagrams
- Design evaluation
- Design patterns
- Software architecture

DePaul University

What is this course about?

- **Object-Oriented Design Principles**
  - You will have a deeper understanding of object-oriented concepts and how to use them, and will be able to design and develop software applications using object-oriented design principles.
- **Visual Modeling**
  - You will be able to model a software solution visually using UML sequence and class diagrams.
- **Design Patterns**
  - You will be able to design and implement an executable solution to a given problem in a programming language using the most suitable set of common software and architectural design patterns.
- **Object-Oriented Programming Paradigm Principles**
  - You will be able to effectively translate design patterns and object-oriented design principles into an object-oriented programming language.

DePaul University

What is this course about?

- Not just programming
  - Designing programming solutions
  - What does it mean ?
- Common approaches
  - Patterns
- School Vs. Real World
- Writing maintainable, extensible high quality code

DePaul University

Goal?

- This course is about taking advantage of Object-Oriented concepts to write maintainable, extensible code

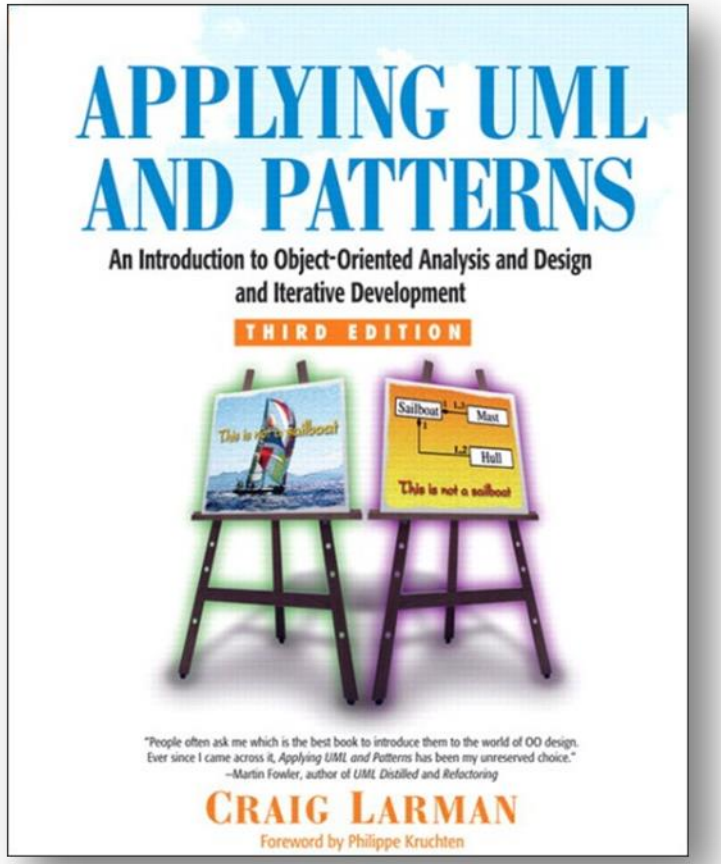
DePaul University

Prerequisite and References

- Prerequisite :
  - Data Structures II (CSC 301, 403, 383, 393), or Similar course
  - Some experience programming in ***Java*** or another ***C-like*** language is required. This is **NOT** an introductory Java course.

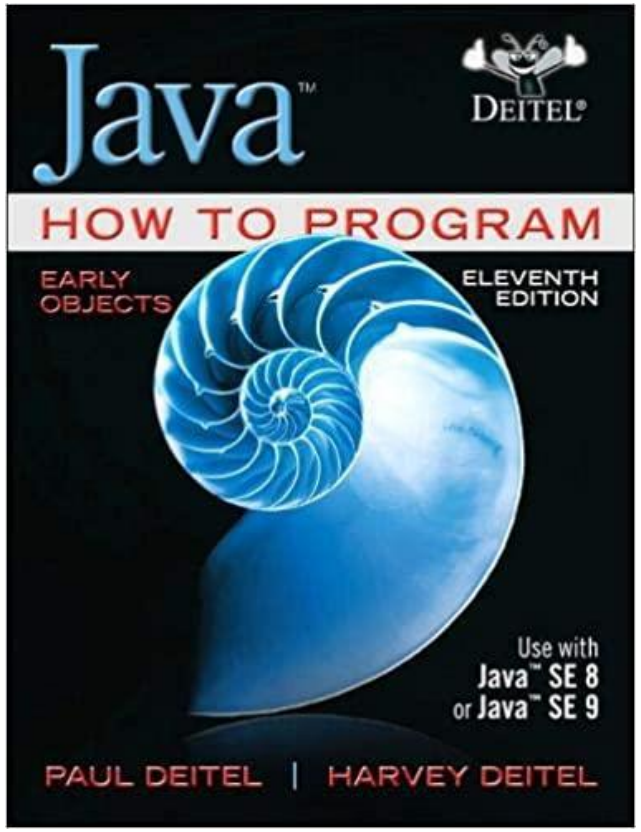
References (not required)

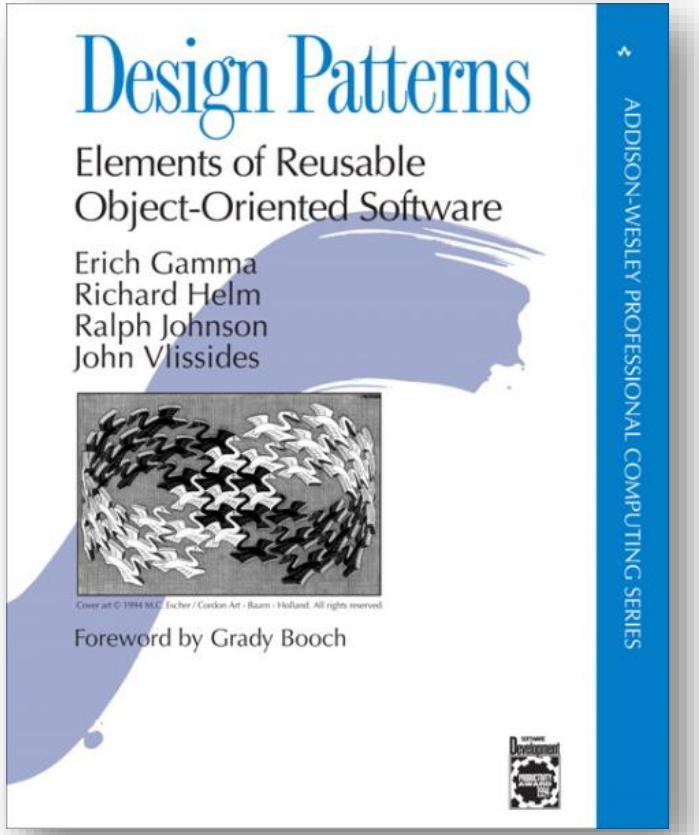
Design

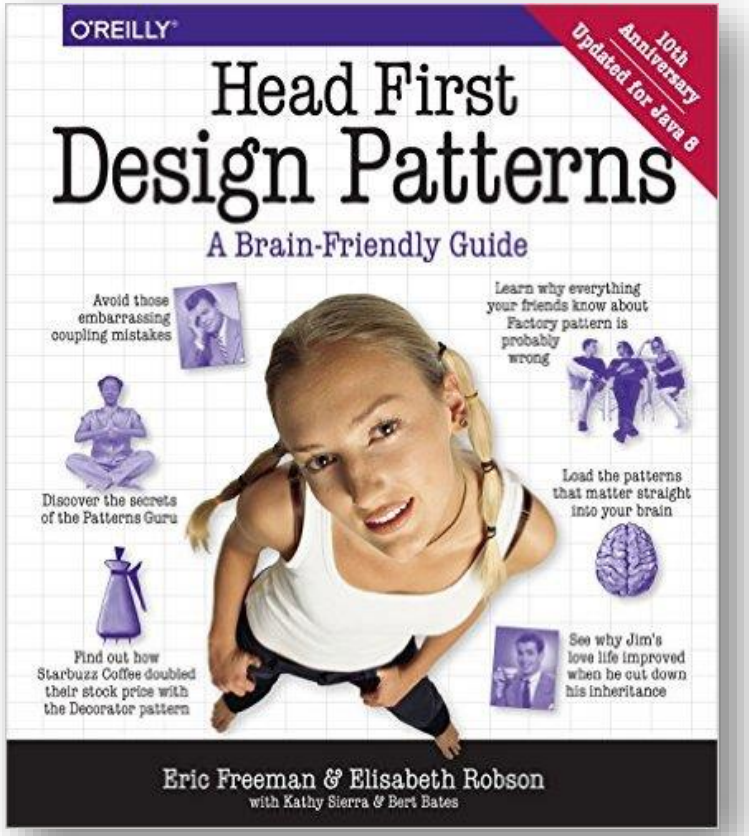


DePaul University

Java and Design Patterns







Assignments and Project

- Assignments:
  - 3 Assignments -- 35%
  - Must be submitted via Desire2Learn by 11:59 PM **Chicago Time** on the assignment due date.

- Design Pattern Presentation:
  - One presentation + Source code -- 15%
  - Each 2 students will choose a design pattern (topics NOT covered in the class)

  - Late submissions will not be accepted unless with a prior approval from the instructor.

Exams and Assessment

- **Exams**
  - Mid-term exam -- 20%
  - Final exam -- 30%

Tentative Schedule

Week	Lecture Topics	Assignment Release date
09/13/2021	• Introduction	
09/20/2021	• OOP Principles and UML part 1	
09/27/2021	• UML part 2	Assignment 1
10/04/2021	• Design to code and Introduction to Design patterns	
10/11/2021	• Midterm exam	
10/18/2021	• Design Patterns 1	Assignment 2
10/25/2021	• Design Patterns 2	
11/01/2021	• Design Patterns 3	
11/08/2021	• Design Patterns 4	Assignment 3
11/15/2021	• Anti pattern/Student Presentations	Presentations
11/22/2021	• Final Exam	

DePaul University

Outline

- Introduction
- Syllabus
- Introduction to Object Oriented Software Development

Software Engineering

- Software engineering entails the design and implementation of the software systems on which our society depends:

- Building
- Biotechnology
- Games
- Government
- Sports
- Submarines








DePaul University

Software Engineering

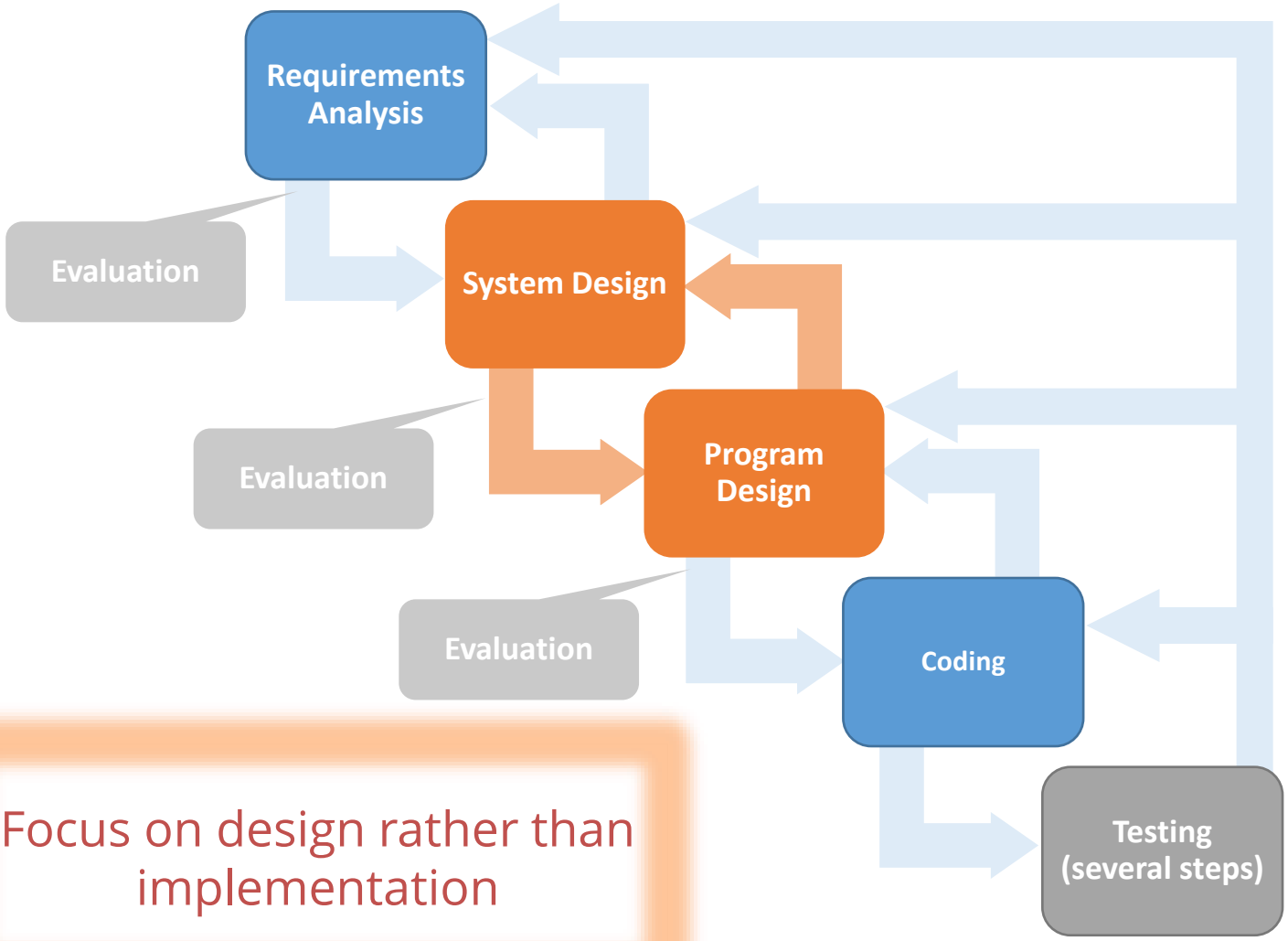
- Software engineering is all around you
  - Google, Facebook, Twitter, Youtube
    - Have to sustain a large number of visitors, maintain massive amounts of storage.
  - Games
    - Develop software to enable sophisticated graphics, rendering, etc.
  - Mobile devices
    - Application to be developed by software engineers.

Software development activities

- **Domain modelling** : modelling what already exists in the domain
- **Requirements** : identifying, categorizing, prioritizing and modelling what the system must do
- **Analysis**: modelling the structure and behavior of the system will meet its specification from a user's perspective, moving from the domain to a software solution
- **Design:** Deciding on the distribution of responsibilities to fulfil that specification
- **Implementation:** Producing code that will meet the user requirements
- **Testing and deployment:** ensuring that the software does meet its requirements to give a runnable system

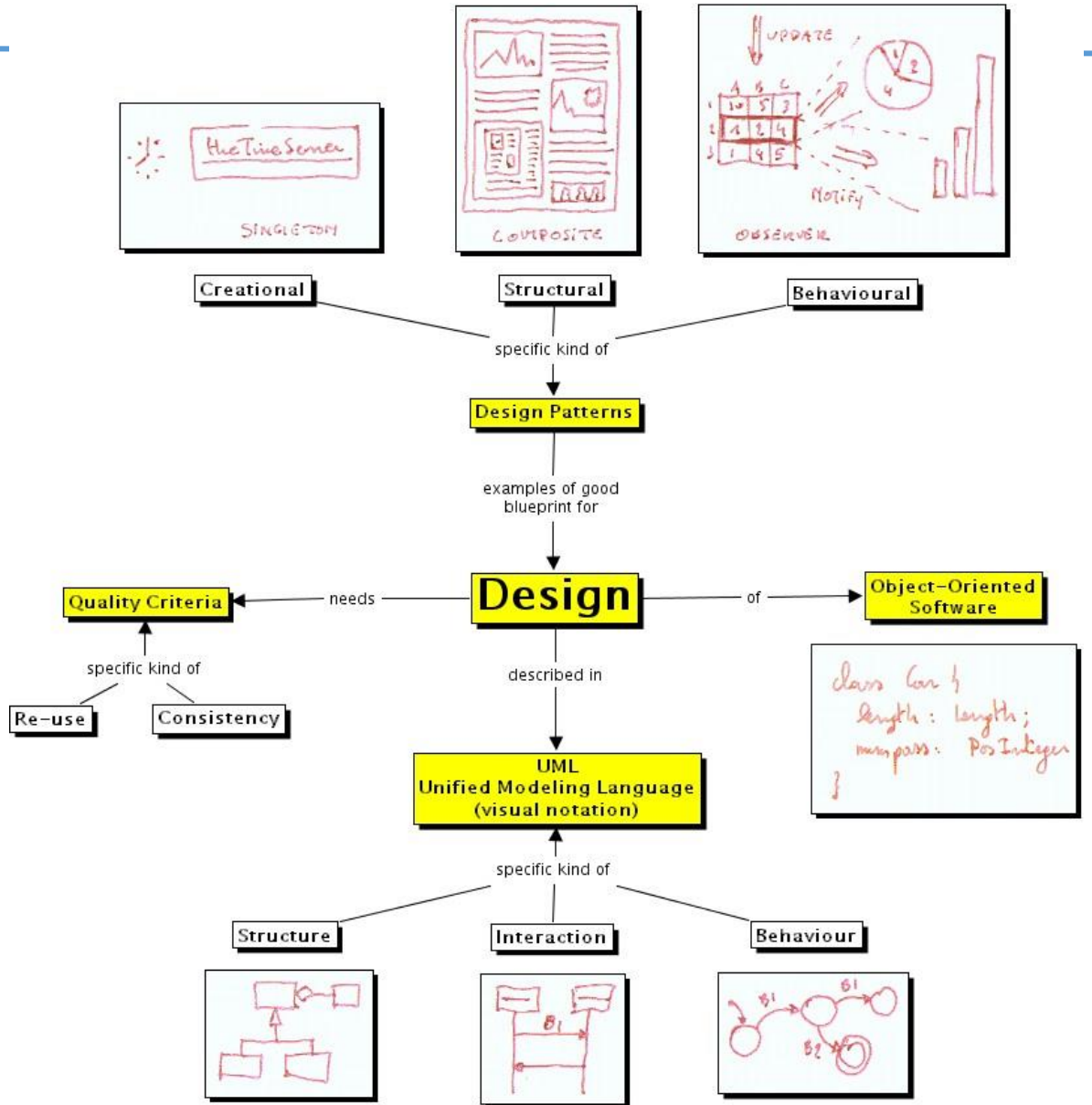
Special attention to design

The most critical step in development



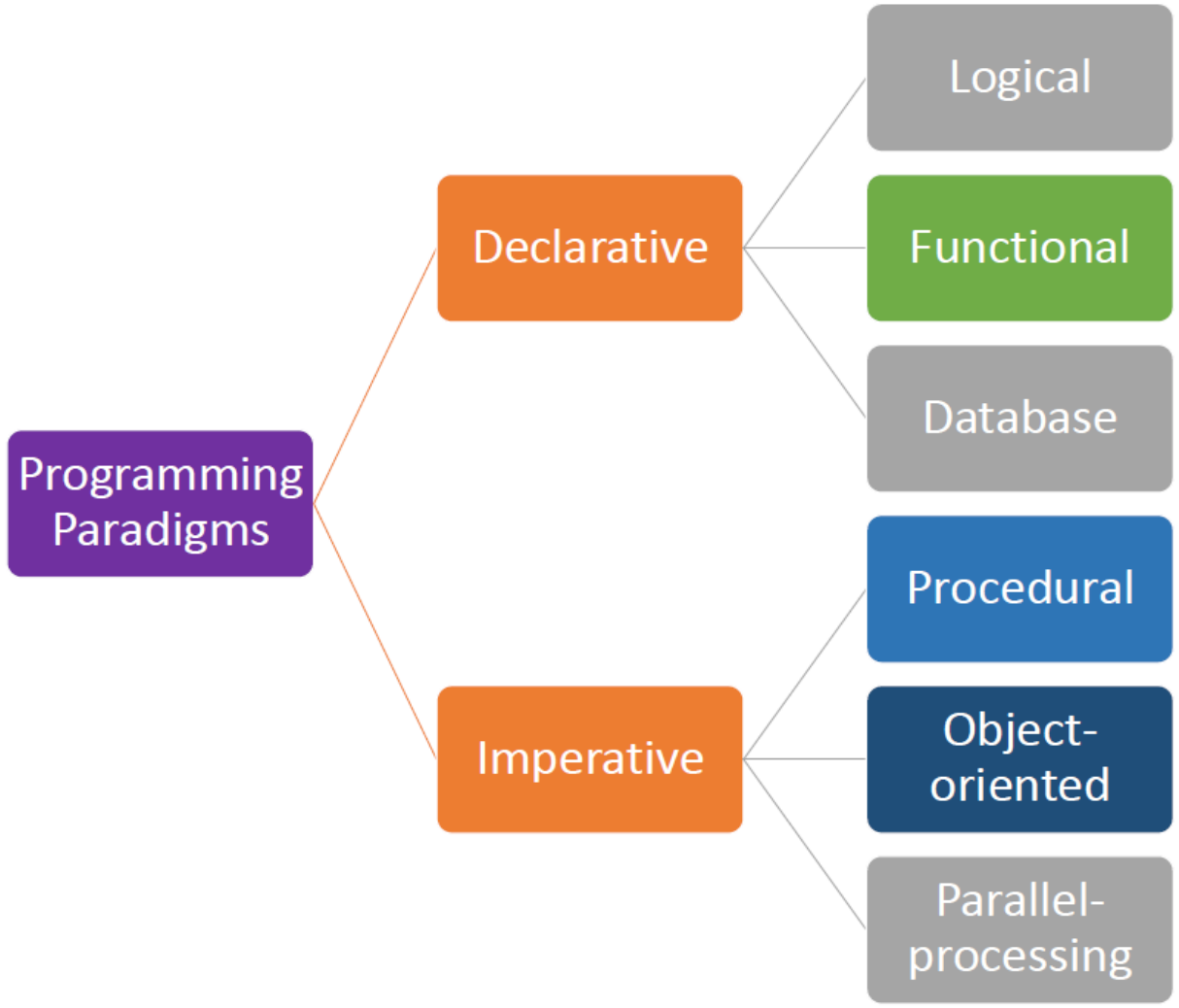
Focus on design rather than implementation

Main components in Software development



Programming Paradigms

- **What is Programming Paradigm?**
  - Imperative
    - HOW to execute program logic
  - Declarative
    - WHAT to execute



DePaul University

Programming Paradigms

- **Imperative programming paradigm**
  - Procedural Programming paradigm
    - C, Pascal
  - Object oriented programming
    - Java, C++, Python
  - Parallel processing approach
    - NESL language

DePaul University

Programming Paradigms

- **Declarative programming paradigm**
  - Logic programming paradigms
    - Prolog
  - Functional programming paradigm
    - Javascript
    - Scala
    - Lisp
  - Database/Data driven programming approach
    - SQL

DePaul University

- Introduction
- Syllabus
- Introduction to Object Oriented Software Development
  - Object-oriented Programming : Basics

Class

- A class is a representation of a type of object.
- A class is the blueprint from which the individual objects are created

```
public class Student
{  }

Student objectStudent = new Student();
```

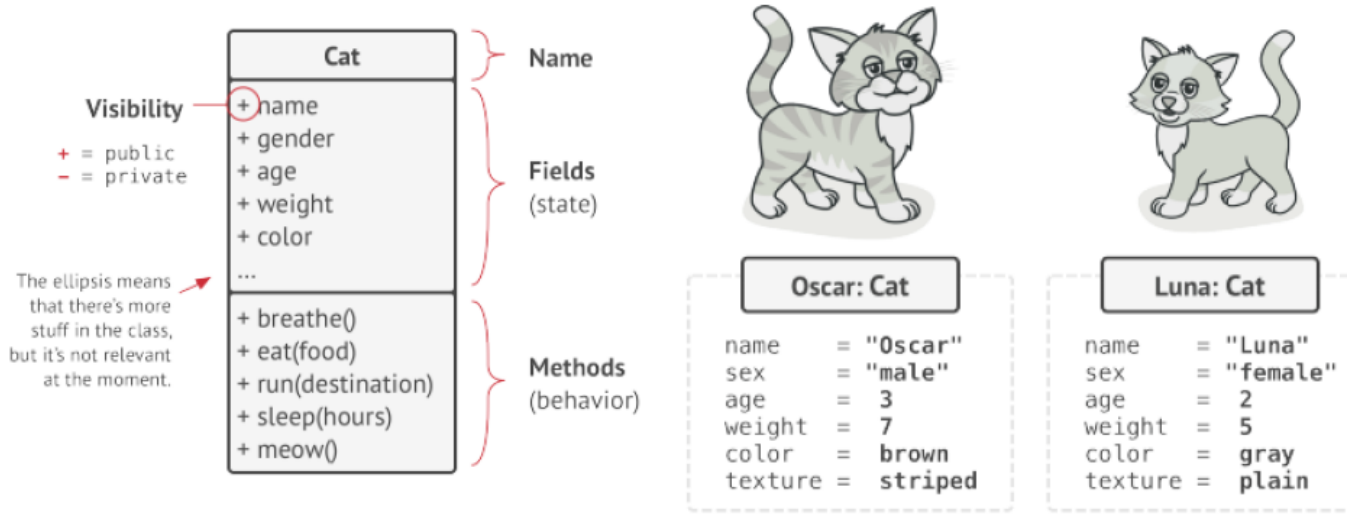
DePaul University

Object

- An object can be considered a “thing” that can perform a set of related activities.
- Set of activities defines the object’s behavior
- Example :
  - Hand (object) can grip something
  - Student can give their name or address
- Object = instance of a class

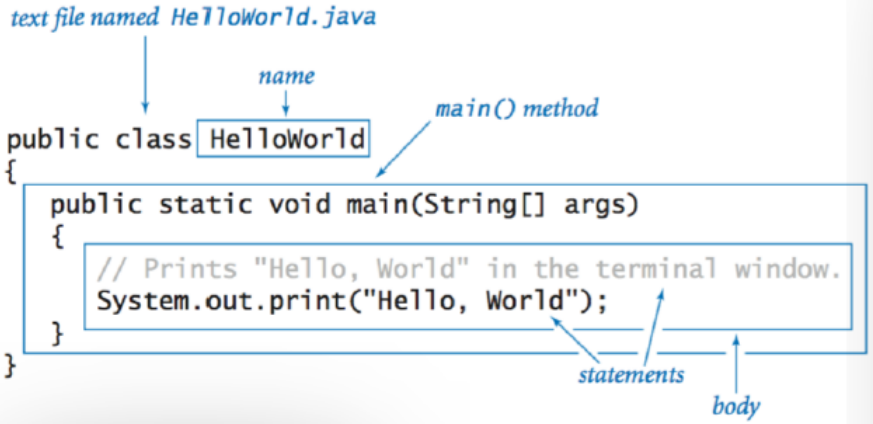
OOP Building Blocks : Classes and Objects

- **Class**
  - Group of objects
  - User defined data types
- **Object**
  - Object = Instance



OOP Building Blocks : Classes and Objects

- **Class**
  - How to create a Class?
  - structure of a class
- **Object**
  - How to create an object?
    - Declaration
      - Creating a *reference*
      - No memory allocation
    - Instantiation
    - Initialization




```
type objectName;
```

```
//Example of Initialization and Instantiation on the same line
SomeClass s; // Declaration
s = new SomeClass(); // Instantiates and initializes the memory and initializes the variable 's'
```

```
//Example of Initialization of a variable on a different line to memory
void someFunction(SomeClass other) {
    SomeClass s; // Declaration
    s = other; // Initializes the variable 's' but memory for variable "other" was set somewhere else
}
```

```
//The Structure of a Java Class
class Car { // Class name

  // Class Data members
  int topSpeed;
  int totalSeats;
  int fuelCapacity;
  String manufacturer;

  // Class Methods
  void refuel(){
    ...
  }
  void park(){
    ...
  }
  void drive(){
    ...
  }

  // Main method
  public static void main(String args[]) {
  ClassName obj = new ClassName(); // className object
}
```

OOP Building Blocks: Attributes & Methods

- **Attributes**
  - object's state
- **Methods**
  - object's behavior
  - Method parameters
  - Method Overloading

```
class Car {

  // Public method to print speed
  public void printSpeed(int speed) {
    System.out.println("Speed: " + speed);
    }

}

class Demo {

  public static void main(String args[]) {
    Car car = new Car();
    car.printSpeed(100); // calling public method
  }

}
```

```
class Demo {

   public static void main(String args[]) {
     Car car = new Car();
     car.setSpeed(100); // calling the setter method
     System.out.println(car.getSpeed()); // calling the getter method
   }

}
```

OOP Building Blocks: Constructors

- **Constructor**
  - Initializing new object states
  - Can be overloaded
  - No return type
  - Only called once
- **2 Types**
  - Default / non-parameterized
  - Parameterized

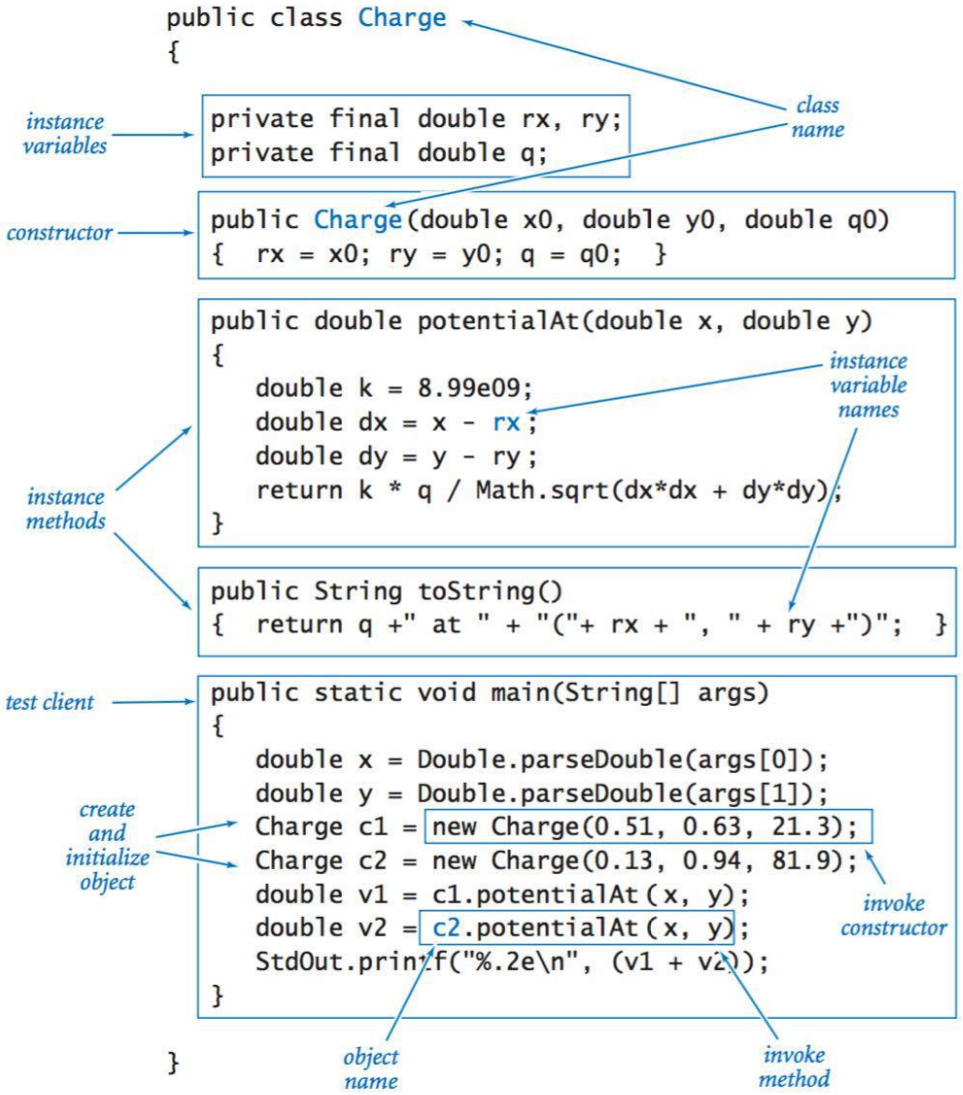
```java
class Date {

  private int day;
  private int month;
  private int year;


  // Default constructor
  public Date() {
    // We must define the default values for day, month, and year
    day = 0;
    month = 0;
    year = 0;
  }

  // Parameterized constructor
  public Date(int d, int m, int y){
    // The arguments are used as values
    day = d;
    month = m;
    year = y;
  }

  // A simple print function
  public void printDate(){
    System.out.println("Date: " + day + "/" + month + "/" + year);
  }
}

class Demo {

  public static void main(String args[]) {
    // Call the Date constructor to create its object;
    Date paramDate = new Date(1, 8, 2018); // Object created with specified values!
    Date defaultDate = new Date(); // Object created with default values!
    paramDate.printDate();
    defaultDate.printDate();
  }
}
```

OOP Building Blocks: Constructors

```
A obA=new A();
```

```
class A
{
  A()
    {
      //some code
    }
}
```

```
class A
{
  public A()
  {
    System.out.println("Constructor with no parameter");
  }
  public A(int a)
  {
    System.out.println("Constructor with one integer parameter");
  }
  public A(int a,int b)
  {
    System.out.println("Constructor with two integer parameter");
  }
  public A(double a)
  {
    System.out.println("Constructor with one double parameter");
  }
}
```


public class Charge
{
private final double rx, ry;
private final double q;
public Charge(double x0, double y0, double q0)
{ rx = x0; ry = y0; q = q0; }
public double potentialAt(double x, double y)
{
double k = 8.99e09;
double dx = x - rx;
double dy = y - ry;
return k * q / Math.sqrt(dx*dx + dy*dy);
}
public String toString()
{ return q +" at " + "("+ rx + ", " + ry +")"; }
public static void main(String[] args)
{
double x = Double.parseDouble(args[0]);
double y = Double.parseDouble(args[1]);
Charge c1 = new Charge(0.51, 0.63, 21.3);
Charge c2 = new Charge(0.13, 0.94, 81.9);
double v1 = c1.potentialAt(x, y);
double v2 = c2.potentialAt(x, y);
StdOut.printf("%.2e\n", (v1 + v2));
}
}
instance variables
class name
constructor
instance methods
instance variable names
test client
create and initialize object
invoke constructor
object name
invoke method

Examples (1)

- Example Code 1[oopBasics1]
- Q&A
  - Constructor return type ?
- Example Code 2 [oopBasics1]
- Q&A
  - Why does the compiler give error? Shouldn't I have a default constructor?

Source code examples that we are going through the lectures of this course are available on D2L after each lecture

DePaul University

Examples (1)

- Example Code 3 [package oopBasics1]
- Constructor Overloading example
- Q&A
  - What is *this* keyword?
- Local variable
  - Declared inside methods, blocks, or constructors
- Instance variable
  - Declared inside a class but outside a method, block, or constructor

DePaul University

- Introduction
- Syllabus
- Introduction to Object Oriented Software Development
  - Object-oriented Programming : Basics
  - Object-oriented Programming : In-Depth

Static Methods and Variables

- *Static* keyword
- Static : can be accessed before any objects of its class are created, and without reference to any object
- Static member = Class member
- Common to all instances of the class
- Example [package oopBasics2]
  - Defining and accessing class members

DePaul University

- Access modifiers
  - Public, private, protected, default
  - public > protected > package-private (or default) > private
- Public vs. Private
- Q&A
  - Why Main() method is always public?
  - What happens when no access modifier is defined?
- Example (Access Control) [package oopBasics2]
  - Accessing the public and private members

DePaul University

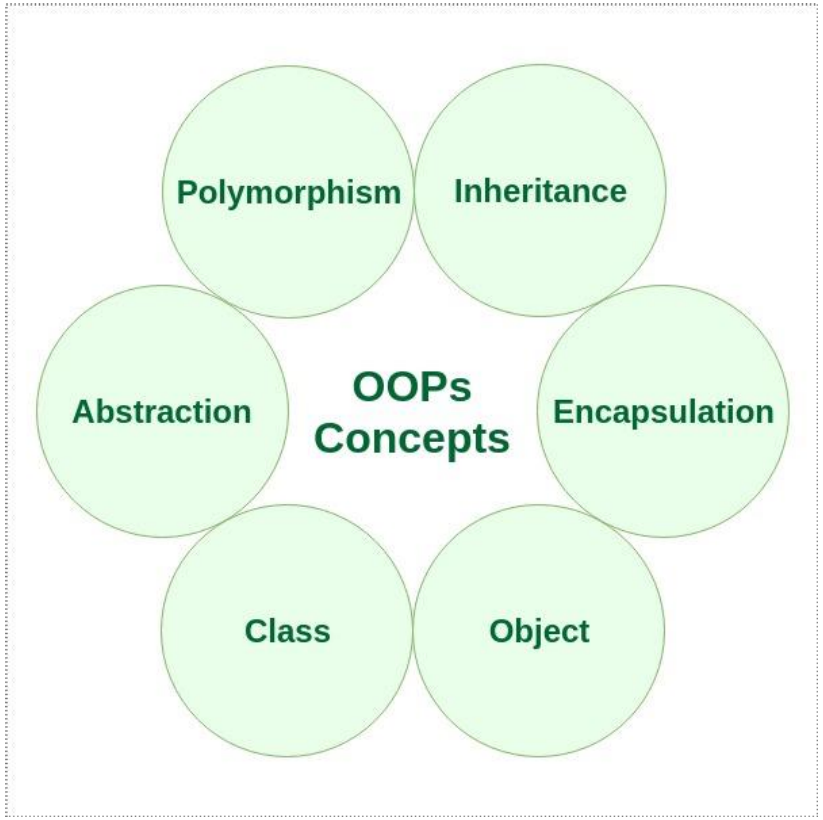
Getters and Setters

- How to access private members of a class ?
  - Public getter and setter methods
- Example [oopBasics]
  - Creating getter and setter
  - Generate with IDE
  - Generate using lombox Project
    - https://projectlombok.org/
- Q&A
  - `What are the benefits of using Getter & Setter?

Concepts

- A software system may consist of many classes
  - Need to be managed

- Four main concepts
  - Encapsulation
  - Abstraction
  - Inheritance
  - Polymorphism



DePaul University

- Questions ?