CS 417
Design Patterns

Motivation and UML Modeling

Dr. Chad Williams
Central Connecticut State University

Topics

- Motivation
  - Challenges
  - Development activities
  - Object orientation
  - Quality system
- UML

Motivation

- Software is backbone of all major services
- Initial design/development occur
- Maintenance and enhancement cycle dominates cost & time
  - Requirements change
  - Functionality added
  - Error correction
  - Each change has the potential to make the system more fragile
- Errors can cause serious issues
  - 1990, AT&T had 8+ hour outage due to misplaced `break` statement in C program
  - 1996, communication satellite exploded on liftoff due to number conversion error
  - 2001, software error caused NYSE to shutdown trading floor for over an hour

Challenges

- Complexity
- Longevity and evolution
- High user expectations

- Iterative development compresses these

Development activities

- Requirements analysis
- Design
- Implementation & unit testing
- Integration & system testing
- ***Maintenance***

Goals of software systems

- Usefulness
- Timeliness
- Reliability
- Maintainability
- Reusability
- Efficiency
- User friendliness

Factors of maintainability

- Flexibility
- Simplicity
- Readability

- How does object oriented paradigm attempt to address each of these?

Iterative development

- Conceptualization
  - Establish core requirements for **this iteration**
- OO analysis and modeling
- OO design
- Implementation and testing

- **Critical to refactor and look for emerging places where patterns make sense each iteration**

OO analysis and modeling

- Build models of system's desired behavior
  - Identify relevant aspects with real world
  - Define services provided/problems to be solved
- Model is simplification of reality used to better understand the system
- UML used to create use cases, class diagrams, etc. to describe these relationships precisely

Object-oriented design

- Goal – create architecture for implementation
- Represent objects, classes and relationships among them
- Purpose of design ensure:
  - Satisfy requirements and provide desired services
  - Design flexible enough to accommodate future changes and enhancements
  - Feasible and efficient

Implementation

- Converting design into working program
- Involves coding, unit testing, debugging
- Concerns include
  - Is implementation correct?
  - Is implementation efficient and maintainable?
  - Is implementation robust, tolerate faults and recover from failures?

Maintenance

- Goal - manage post-delivery evolution
- Maintenance tasks
  - Removing bugs
  - Enhancing functionality
  - Adapting system to evolving needs and environments

Key features of iterative development

- Each iteration relatively small can be completed in short period of time
- Each iteration results in a release of an **executable** product or component, which is part of the final product
- Must try to avoid the temptation to only add on. Refactoring to take advantage of design patterns is critical to avoiding code becoming increasingly fragile and greatly improves the maintainability and ease at which enhancements can be made in the future.

Rational Unified Process (RUP)

- Guidelines for carrying out development activities
- Key practices
  - Iterative development
  - Systematically elicit, organize, and manage changing requirements
  - Component based architecture
  - Visually model using UML
  - Continuously verify software quality
  - Control changes to software

RUP

- Build models rather than paper documents
  - Business model: abstraction of organization
  - Domain model: establishes context
  - Use case model: functional requirements
  - Analysis model (optional): idea design
  - Design model: core model for driving implementation mapping models to solutions
  - Process model (optional): concurrency and synchronization
  - Deployment model: hardware topology
  - Implementation model: parts used to assemble release of physical system
  - Test model: paths to validate and verify system

We will cover some aspects of this

Modeling using UML

Topics

- Principles and concepts
- Modeling relationships and structures
- Modeling requirements with use cases

Unified Modeling Language (UML)

- Graphical notation for describing OO models
- Provides standards used across projects
- Notation is precise and descriptive
- Allows visual models to largely replace text based documentation

Class vs Object

- Both can be viewed in terms of:
  - Interpretation of real world, and
  - Representation in model

	Interpretation in real world	Representation in model
Class	Represents a set of objects with similar characteristics and behavior	Characterizes the structure of states and behaviors shared by all instances
Object	Represents anything in real world that can be distinctly identified. Instances of class	Has an identity, a state, and a behavior

Class

- Defines template for creating instances/objects
- Defines attributes, methods, behaviors
  - Names and types of all attributes
  - Method names, arguments (name, type), implementation
- Allows all instances to be treated in a uniform fashion

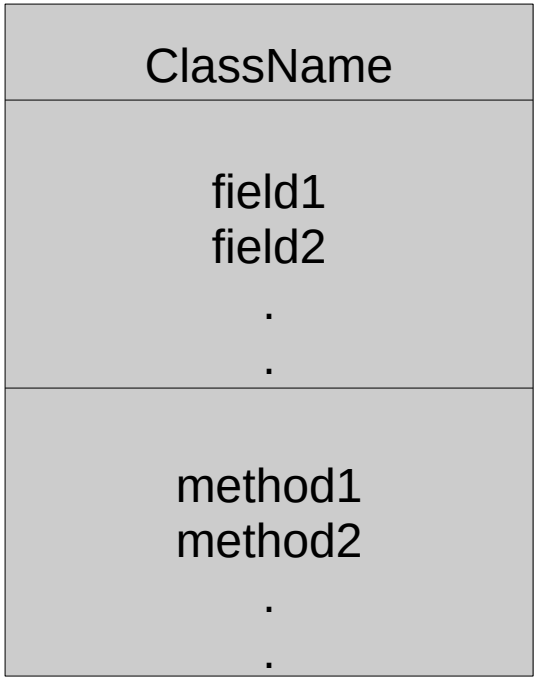
Object

- Object and instance used interchangeably
- Represent a specific instance of its class
  - Ex. Chad is an instance of Person
- Each object has its own set of attribute values or state

Terminology

- Objects are **equal** if states are the same
- Objects are **identical** if refer to same object
- **Accessor** is a method that allows you to read the state of an object (does not change the state)
- **Mutator** is a method that changes the state of an object
- **Immutable object** is an object whose state may never be modified by any of its methods – i.e. state is constant

UML notation for classes



UML notation cont.

- Syntax of variable names is:
  - [visibility]name[[multiplicity]][type][=initial value]
- Syntax of methods is:
  - [visibility] name([parameters]) [type]

Visibility

- **Public** – Feature is accessible to any class
- **Protected** – Feature available in the class itself, all classes in its same package, and all its subclasses
- **Package** – Feature is accessible to the class and all classes in the same package
- **Private** – Feature is only accessible within the class itself

Language syntax

Keyword	C#	C++	Java
private	class	class	class
protected internal	same assembly and derived classes	-	-
protected	derived classes	derived classes	derived classes *and/or* within same package
package	-	-	within its package
public internal	same assembly	-	-
public	everybody	everybody	everybody

UML notation visibility

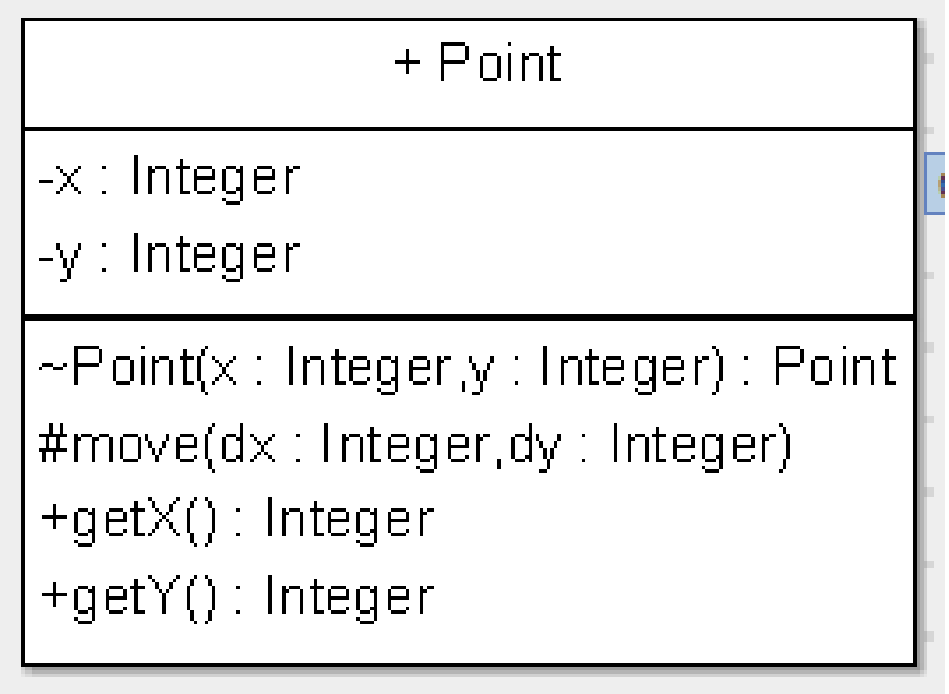
Visibility	Java syntax	UML syntax
public	public	+
protected	protected	#
package		~
private	private	-

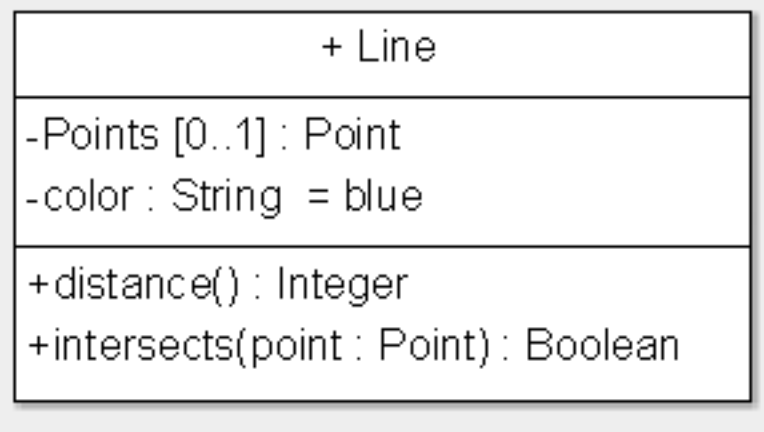
Simple Java class

```
public class Point {
   private int x;
   private int y;
   Point(int x, int y){}
   protected void move(int dx, int dy){}
   public int getX(){return x;}
   public int getY(){return y;}
}
```

Visibility will be key!

UML of Point class





Group work

- Split into groups 3 to 4 people
- Make up 3 classes for 2D geometric shapes. Draw UML for them, particularly think of attributes and methods that would require different visibilities – think moving, reshaping/manipulation, calculations, checks

Visibility	Java syntax	UML syntax
public	public	+
protected	protected	#
package		~
private	private	-