

CSC 207: Object-oriented Programming, Problem Solving, and Data Structures

Guest Lecture: Object-oriented Modeling

William Rebelsky
[rebelskyw]

Grinnell College

Wednesday 29th January, 2025

Reminders

- There is no class on Friday (faculty retreat), however your instructor may have assigned a reading or homework
- Reading 2.1 (Program Reasoning Tools) and the associated daily drill are due before class on Monday
- CS Extras Tomorrow January 30th: Declaring a CS Major. If you are interested in declaring, you should go to the session to learn about the unique process for major declarations in the CS department

Notes

- The lecture today will be based on the readings provided by Professor Peter-Michael Osera
- Reading 1.2: Using Objects
- Reading 1.3: Object-oriented Modeling

Table of Contents

1 Using Objects (Review from Monday)

2 Object-oriented Modeling

What is an Object?

What is an Object?

- Definition: An object possesses *state* and *behavior*
- State:
 - Fields or instance variables
- Behavior:
 - Methods

Client Perspective on Objects

Three Fundamental operations:

Client Perspective on Objects

Three Fundamental operations:

- ❶ Create or *instantiate* objects
 - *New Expressions*: `new <class name>( <parameters> )`
- ❷ Access the state or *instance variables* of objects
 - *Dot Notation*: `<object>.<field name>`
 - Generally not good practice (Abstraction)
- ❸ Invoke the behavior or *methods* of objects
 - *Dot Notation*: `<object>.<method name>( <parameters> )`

Table of Contents

1 Using Objects (Review from Monday)

2 Object-oriented Modeling

Anatomy of a Class

Classes are sort of like a swiss-army knife in object-oriented languages like Java; they do many things:

Anatomy of a Class

Classes are sort of like a swiss-army knife in object-oriented languages like Java; they do many things:

- Classes act as a blueprints for objects
- Classes are the way of defining user-defined types in Java
- Classes act as a namespace for collections of static functions and variables
- Classes act as an abstraction mechanism separating features from implementation through interfaces

Anatomy of a Class

Classes are sort of like a swiss-army knife in object-oriented languages like Java; they do many things:

- Classes act as a blueprints for objects
- Classes are the way of defining user-defined types in Java
- Classes act as a namespace for collections of static functions and variables
- Classes act as an abstraction mechanism separating features from implementation through interfaces

An *object* is an *instance* of a *class*

Class Declaration Syntax

```
<visibility modifier> class <class name> {  
    <field and method declarations>  
}
```

- ❶ Field Declarations: <visibility modifier> <type> <name>;

Class Declaration Syntax

```
<visibility modifier> class <class name> {  
    <field and method declarations>  
}
```

❶ Field Declarations: <visibility modifier> <type> <name>;

❷ Method Declarations:

```
<visibility modifier> <type> <name>( <arguments> ) {  
    <statements>  
}
```

Class Declaration Syntax

```
<visibility modifier> class <class name> {  
    <field and method declarations>  
}
```

❶ Field Declarations: `<visibility modifier> <type> <name>;`

❷ Method Declarations:

```
<visibility modifier> <type> <name>( <arguments> ) {  
    <statements>  
}
```

❸ Constructor Declarations: Similar to method declarations with two changes:

- The name of the method is the name of the class
- there is no return type

“this” Keyword

“this” is an expression that evaluates to the object that is the subject of the constructor or method call

```
public Student (String firstName, String lastName, int age) {  
    this.firstName = firstName;  
    this.lastName  = lastName;  
    this.age       = age;  
}
```

Static

- “static” differentiates between static functions and methods
- static Functions:
 - use “static”
 - tied to the overall class
- Methods:
 - don’t use “static”
 - tied to each object (instance of the class)

Thinking with Objects

Object-Oriented Programming: Solve problems by decomposition into classes that should:

Thinking with Objects

Object-Oriented Programming: Solve problems by decomposition into classes that should:

- Be as small and as simple as possible
- Be limited in scope, ideally, serving only a single distinct purpose in the overall program
- Function as independent units (as much as possible). This allows us to reason about the packages for correctness independently, either during debugging or testing.
- Possess a well-defined interface with clear guarantees about inputs and outputs
- Be able to hide the details of the package that are unnecessary for clients of the package to know about, i.e., its implementation details

Thinking with Objects

- Encapsulation: bundling data and behavior into packages that satisfy the prior criteria
- In Java: classes
- Caveat: We as programmers need to enforce the properties through good object-oriented design principles

Thinking with Objects

- Encapsulation: bundling data and behavior into packages that satisfy the prior criteria
- In Java: classes
- Caveat: We as programmers need to enforce the properties through good object-oriented design principles
- First three properties: Classes should be *small* and *specific*
- Last two properties: *Abstraction*. Hide the implementation through an interface

Abstraction

- public vs private
 - public: member is visible to everyone
 - private: member is only visible to the class containing the member
- getter/setter methods: all the functionality of public fields without exposing the underlying field/implementation

Javadocs

```
/**
 * Sets the value of the counter.  This value must be non-negative.
 *
 * @param value the new, non-negative value the counter.
 * @throws IllegalArgumentException if a non-negative value is given.
 */
public void setValue(int value) { /* ... */ }
```

- @param <name> <description>
- @return <description>
- @throws

Lab

The rest of class will be Lab work:

- Please work with your partner, ensuring that both of you understand what you are doing before moving on
- Please trade off driver (coding) and navigator (review/feedback/guidance) positions throughout the remaining class
- The lab is available at:
<https://rebelskyw.cs.grinnell.edu/csc-207/>
- I believe you will need to submit your Counter class file to gradescope