Object-Oriented Fundamentals

CSX3002/ITX2001 Object-Oriented Concepts and Programming
CS4402 Selected Topic in Object-Oriented Concepts
IT2371 Object-Oriented Programming

Kwankamol Nongpong, Ph.D.
Assumption University

OO Terminologies

- Class
- Field (Variable)
  - Static field (Class variable)
  - Non-static field (Instance variable)
- Method

Class

What is a class?

Declaring Classes

Access Modifiers

Declaring Classes

- Class declaration is an example of type, a template for
  - a set of data values
  - and the operations that can be legally performed on these values.

```
class CheckingAccount
{
 // field, method,
     // and other member declarations
}
```

Identifier

- Java and many other programming languages are case-sensitive.
  - The case of the letter does matter.
- Java Identifiers consist of
  - letters (**A** -**Z** , **a** -**z** , or equivalent uppercase/lowercase letters in other human languages),
  - digits (**0** -**9** or equivalent digits in other human languages),
  - connecting punctuation characters (such as the underscore),
  - and currency symbols (such as the dollar sign).
- Identifiers can only begin with a letter, a currency symbol, or a connecting punctuation character.

Which one is valid?

- temperature
- Temperature
- door^color
- First$name
- _userid

Reserved Words

- Some identifiers are reserved for special uses:
  - `abstract`, `assert`, `boolean`, `break`, `byte`, `case`, `catch` and many more.

Class Members

- Fields
- Constructors
- Methods

Access Modifiers

- Indicate how a field, method, or constructor can be accessed.
- For now, we will discuss:
  - Public: accessible from anywhere.
    - Classes can be declared public as well.
  - Protected: accessible from all classes in the same package as the member's class as well as subclasses of that class, regardless of package. (More later)
  - Private: cannot be accessed from beyond the class in which it is declared.

Access Levels for Each Modifier

Modifier	Class	Package	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	N
no modifier	Y	Y	N	N
private	Y	N	N	N

Field

What is a field? / Declaring Fields

Static Fields / Read-only Fields

Variables

- Class Variables
  - Variables used in the class context
  - Attributes that are shared by all created objects
- Instance Variables
  - Attributes that are distinct to individual objects

What is a field?

- Variables declared in the class body

Declaring Fields

- Syntax: `type fieldName;`

```
class CheckingAccount
{
  String owner; // name of person who owns
                // this checking account
  int balance;  // amount of money that can be withdrawn
}
```

Primitive Types

Primitive Type	Reserved Word	Size	Min Value	Max Value
Boolean	`boolean`	--	--	--
Character	`char`	16-bit	Unicode 0	Unicode $2^{16} - 1$
Byte integer	`byte`	8-bit	-128	+127
Short integer	`short`	16-bit	-2^{15}	$+2^{15} - 1$
Integer	`int`	32–bit	-2^{31}	$+2^{31} - 1$
Long integer	`long`	64-bit	-2^{63}	$+2^{63} - 1$
Floating-point	`float`	32–bit	IEEE 754	IEEE 754
Double precision floating-point	`double`	64-bit	IEEE 754	IEEE 754

Array-based Field

- Field could be of array types.

```
class WeatherData
{
  String country;
  String[] cities;
  double[][] temperatures;
}
```

Field Default Values

- Each field is initialized to a default value.
  - Fields of type int are initialized to 0.
  - Fields of type String are initialized to null.
- Fields of primitive types are initialized to their default values (see the next slide).
- Fields of reference types are initialized to **null**.

Default Values for Fields

Data Type	Default Value (for fields)
byte	0
short	0
int	0
long	0L
float	0.0f
double	0.0d
char	'\u0000'
boolean	false
String (or any object)	null

Initializing Fields: An Example

```
class WeatherData
{
  String country = "United States";
  String[] cities = {"Chicago",
          "New York",
          "Los Angeles"};
  double[][] temperatures = {{0.0, 0.0},
                {0.0, 0.0},
                {0.0, 0.0}};
}
```

It is not necessary to declare fields at the beginning of the class definition, although this is the most common practice.

It is only necessary that they be declared and initialized before they are used.

Initializing Fields

- This works well when the initialization value is available and the initialization can be put on one line.
- However, this form of initialization has limitations because of its simplicity.
- If initialization requires some logic (for example, error handling or a for loop to fill a complex array), simple assignment is inadequate.
- Instance variables can be initialized in constructors, where error handling or other logic can be used.

Static Field

- Sometimes you need a single copy of a field no matter how many objects are created.
- Just create a static field.
- It is a field that associates with a class instead of with that class's objects.

Scenario

- Suppose you want to track the number of `CheckingAccount` objects that have been created.
- What will you do?

Static Field Usage

- You introduce a counter field into this class.
  - What's the proper initial value?
- You also place code in the class's constructor to increase counter's value by 1 when an object is created.
- However, because each object has its own copy of the counter field, this field's value never advances past 1.

Static Field: An Example

```
class CheckingAccount {
  String owner;
  int balance;
  static int counter;
}
```

What's the initial value for each field?

Read-Only Field

- Specified by a reserved word “final”

```
class Employee {
  final static int RETIREMENT_AGE = 65;
}
```

Method

What is a method?

Declaring Methods

What is a method?

- Defines behavior, action or operations on the data (field) of that class.

Declaring Methods

- Syntax:

```
returnType methodName(parameter list) {
   // method body
}
```

Method: An Example

```
class CheckingAccount {
  String owner;
  int balance;
  static int counter;
  void printBalance()
  {
    // code that outputs
    // the balance field's value
  }
}
```

Exercise: Method Declaration and Implementation

- Add the following methods to CheckingAccount class.
  - deposit
  - withdraw
  - printBalance

Exercise: Method Declaration and Implementation (1)

```
/**
 * Add the specified amount to balance
 * @param amount
 * @return the new balance
 */
int deposit(int amount)
```

Exercise: Method Declaration and Implementation (2)

```
/**
 * Subtract the specified amount from the balance.
 * @param amount
 * @return the new balance
 */
int withdraw(int amount)
```

Exercise: Method Declaration and Implementation (3)

```
/**
 * Print the current balance using the standard
 * currency format i.e, ###,###.## for positive
 * values and (###,###.##) for negative values
 */
void printBalance()
```

Javadoc

Reading Javadoc

Reading Javadoc

- java.util.Scanner

Javadoc Comments

- Javadoc recognizes special comments
  / .... */**
- Javadoc comments are highlighted blue by default in Eclipse
  - Normal comments are highlighted green
- Javadoc allows you to attach descriptions to
  - classes
  - constructors
  - fields
  - Interfaces
  - and methods
- Just place Javadoc comments directly before their declaration statements.

Javadoc Tags

- ***Tags*** are keywords recognized by Javadoc which define the type of information that follows.
- Tags come after the description (separated by a new line).

Javadoc Tags

- Here are some common pre-defined tags:
  - ***@author [author name]*** - identifies author(s) of a class or interface.
  - ***@param [argument name] [argument description]*** - describes an argument of method or constructor.
  - ***@return [description of return]*** - describes data returned by method (unnecessary for constructors and void methods).
  - ***@exception [exception thrown] [exception description]*** - describes exception thrown by method.
  - ***@throws [exception thrown] [exception description]*** - same as ***@exception***.

* @exception and @throws will be discussed later.

Javadoc: An Example

```
/**
 * @author
 */
class CheckingAccount {
  String owner;
  int balance;
  static int counter;
  void printBalance() {
    // code that outputs the balance field's value
  }
}
```

Javadoc: Another Example

```
/**
 * Add the specified amount to balance
 * @param amount
 * @return the new balance
 */

int deposit(int amount) {
  // code that adds the specified amount
  // to balance, and returns the new balance
}
```

Javadoc: Yet another example

```
/**
 * Subtract the specified amount
 * from the balance.
 * @param amount
 * @return the new balance
 */
int withdraw(int amount) {


}
```

Generate Javadoc in Eclipse

- See demonstration

Constructor

What is a constructor?

What is a constructor?

- Constructor are named blocks of code
  - declared in class bodies
  - construct objects by initializing their instance fields
  - perform other initialization tasks

Constructor

- Special method whose name is the same as the class
- Has no return type
- Could be overloaded

Constructor: An Example

```
class CheckingAccount {
  String owner;
  int balance;
  static int counter;
  CheckingAccount(String acctOwner, int acctBalance) {
    owner = acctOwner;
    balance = acctBalance;
    counter++; // keep track of created
        // CheckingAccount objects
  }
}
```

Constructor: Another Example

```
CheckingAccount(String acctOwner) {
  this(acctOwner, 100);
}
```

What does this mean?

In-Class Exercise 1

- Write a class whose instances represent a dice.

- How would you name a class?
- What are the fields?
- What are the methods?

In-Class Exercise 2

- Write a class whose instances represent a single playing card from a deck of cards. Playing cards have two distinguishing properties: rank and suit.

- How would you name a class?
- What are the fields?
- What are the methods?

In-Class Exercise 3

- Write a class whose instances represent a full deck of cards.

- How would you name a class?
- What are the fields?
- What are the methods?

In-Class Exercise 4

- Write a small program to test your deck and card classes.