Csc-18a Chapter 8 Notes

Overview:

This chapter goes more in depth about classes, set/get methods, and more.

Key Points:

* this keyword
* Classes
  + Class composition
  + Set/get methods
  + Same package classes
  + Static variables, methods, and classes
* enum
* final instance variables
* BigDecimal class

Default Constructor

* When an object has no constructor, a default constructor will be supplied.
  + Each instance variable receives a default value
    - 0 for primitive types
    - False for Booleans
    - null for references
  + They can also be initialized when they’re declared in the class body
    - Same syntax as a local variable
* It also performs another task (Section 9.4.1)

Thrown exception type

* The throw statement can also be used to create a new object of type IllegalArgumentException
* IllegalArgumentException(“custom error message”)
  + Notifies the client code that an invalid argument was passed to the method.
* Ex)
  + If (hour < 0 || hour >=24){

throw new IllegalArgumentException( “hour time was out of range”);

}

* After the exception object is created, the throw statement terminates the method and the exception is returned.

String.format

* Returns a String containing the formatted value
* Format is similar to System.out.printf
  + Uses the printf references
  + Only difference is that format returns a String
* Ex)
  + public String toString(){

return String.format( “%02d:%02d:%02d %s”, hour, minute, second);

}

Date/Time API

* Defined by java.time
* New version for Java SE 8
* Learn more at:
  + Download.java.net/jdk8/docs/api/java/time/package-summary.html

Controlling access to methods

* Public
  + Used for the ‘services’ in the classes
* Private
  + Methods used to accomplish tasks behind the scenes
* Protected
  + See Chapter 9.

this keyword

* Lets the program know that the object is referring to itself
* Use keyword to refer to the object’s instance variables and methods
  + Used when there are multiple variables of the same name
* Ex)
  + private int hour;

private int minute;

private int second;

public SimpleTime (int hour, int minute, int second){

this.hour = hour;

this.minute = minute;

this.second = second;

}

* You typically will not use this explicitly to reference other methods within the current object.

Overloaded Constructors

* Constructors that receive different arguments
* Ex)
  + Public Time2(){

this (0,0,0);

}

Public Time2 (int hour){

this(hour, 0 ,0);

}

Public Time2 (int hour, int minute){

this (hour, minute,0);

}

* Constructors can also declare variables in sequential order as seen above
  + Uses this keyword
  + Only usable for the first line of the constructor
    - Syntax:
      * this (0,0,0);
* The method public Time2(){…} is an example of a no-argument constructor
  + Once one is declared, a default constructor will not be created.

set/get methods

* Typically you want to use this more than public instance variables
* Having these methods allows you to control input and output
  + Allows you to troubleshoot inputs and modify data as needed
    - Aka. **Validity Checking**
* **Predicate methods**
  + Accessor methods that return a true or false

Class Composition

* Classes can references objects of other classes as members
  + Aka. **Has-a relationship**
* An object with another object within its method
* Example: When an alarmclock object receives the current time and time for alarm from Time object

**enum** types

* Like a class, enum types are reference types
* An enum type is declared with an enum declaration
  + A comma-separated list of enum constants
* Each number declaration declares an enum class with restrictions:
  + Implicitly final
  + Implicitly static
  + Any attempt to create an object of an enum type with operator new results in a compilation error
* Ex)
  + public enum book{

//declare constants of enum type

JHTP(“Java How to Program”, “2015”),

CHTP(“C How to Program”, “2013”),

[…]

CSHARPHTP(“Visial C# How to Program”, “2014”);

//instance variables

private final String title;

private final String copyrightYear;

//enum constructor

Book (String title, String copyrightYear){

this.title = title;

this.copyrightYear = copyrightYear;

}

//Get statements

[…]

} //End of enum book class

* Must declare enum constants before the enum type’s constructors, fields, and methods.
* For every enum, the compiler generates the static method **values** that returns the array fof the enum’s constants in the order they were declared.
  + Ex)
    - for (Book book : Book.values()){

//Output values

}

* Static method **range**
  + Declared in **java.util.EnumSet**
  + Displays the range of the enum Book’s constant
  + Receives two parameters:
    - First and last enum constants in the range

Garbage Collection

* To sum it up, Java contains safety measures to terminate memory leaks
* JVM executes garbage collector
* Ex of memory leak:
  + Program ends but the file it opened is still unaccessible since it was not terminated

Class Object’s **finalize** method

* Never use this method
  + It can cause many problems and its uncertain whether it will ever get called before a program terminates.
  + (More in Chp 9)
* It’s intention was to allow the garbage collector to terminate before reclaiming the object’s memory
  + Provides a method to release resources when they’re no longer needed in a program.
* **AutoClosable** objects reduce the likelihood of resource leaks when used with try-with-resources statement
  + An object that is closed automatically once the try-with-resources statement finishes using the object.
  + (More in Section 11.12)

**static** Class Members

* Every object has its own copy of the instance variable in the class
* By declaring a **static** instance variable, one copy of the variable is shared by all objects of the class.
* Theory Ex)
  + Martian objects are meant to battle creatures. If martians < 5, they become scared and run. If martians >=5, they become brave and fight.
  + In this ex, a static instance variable could be used to count every martian without needing to create a counter under every single object.
    - Saves memory
    - More Optimized
* Example on pg. 332
* Use when all objects must use the same copy of the variable.
* Side note:
  + When a static variable is present within an object, the static variable is created regardless whether or not the object is created.

**static** methods

* Used to access **static** variables
* They cannot access a class’s instance variables and instance methods
  + Read side note
    - A **static** method cannot call something that hasn’t been created yet.
  + This results in a compilation error
* Do not use **this** keyword in a **static** method
  + Results in compilation error
* Invoke every static method with the class name and dot(.) operator to emphasize that it is a static method

String object properties

* String objects in Java are *immutable*
  + They cannot be modified after its creation.
* When adding onto a String object, a new object is created containing the values
  + Original is not modified.

**static** import

* Enables you to import the static members of a class
  + Accessible via their unqualified names
    - Classname.staticmethodname();
* Import Forms:
  + Single static import
    - Imports one static member
      * Syntax)
        + import static packageName.ClassName.staticMemberName;
  + Static import on demand
    - Imports all static members
      * Syntax)
        + Import static packageName.ClassName.\*;
  + This allows you to use the methods without preceding the method with the class name
    - Meaning you don’t have to call the Math name when using from the Math class.

**final** instance variables

* It is typically good practice to only allow the user to access and modify values necessary
* Thus if a value should not be changed, use the **final** keyword to make it constant