CompSci 230 S1 2020
Object Oriented Software Development

Classes and Objects: A Deeper Look

Review Quizzes

- What is the output of the following program?

```
public class TestStatic {
  public static int y = 10;
  public TestStatic() {
    y++;
  }
  public TestStatic(int x) {
    y = x++;
  }
  public static void main(String[] args) {
    TestStatic a1 = new TestStatic();
    TestStatic b1 = new TestStatic();
    TestStatic c1 = new TestStatic(100);
    System.out.println(TestStatic.y);
 }
}
```


Review Quizzes

- What is the output after executing the following code?

```
MyPoint p = new MyPoint(10, 20);
p.setX(-1);
System.out.println(p);
```

```
class MyPoint {
    private int x;
    private int y;
    public MyPoint() { … }
    public MyPoint(int x, int y) {…}
    public String toString() {
      return String.format("(%d, %d)", x, y);
    }
    public void setX(int x) {
      if (x>0)
         this.x = x;
    }
}
```


Agenda & Reading

- Topics:
  - Case study: Time class
  - Case study: Time2 class
  - Composition: Employee & Date
- Reading
  - Java how to program Late objects version (D & D)
    - Chapter 7 & 8
  - The Java Tutorial
    - Classes:
      - http://docs.oracle.com/javase/tutorial/java/javaOO/classes.html
    - Objects
      - http://docs.oracle.com/javase/tutorial/java/javaOO/objects.html

Time1.java

1.Case Study: Time1 class

- Class Time1 represents the time of day.
  - **private** int instance variables **hour**, **minute** and **second** represent the time in universal-time format (24-hour clock format in which hours are in the range 0–23, and minutes and seconds are each in the range 0–59).
  - **public** methods **setTime**, **toUniversalString** and **toString**.
  - Class Time1 does not declare a constructor, so the compiler supplies a **default** constructor.
    - Each instance variable implicitly receives the default int value.

```
public class Time1 {
  private int hour;   // 0 - 23
  private int minute; // 0 - 59
  private int second; // 0 – 59

  public void setTime( int h, int m, int s ) { ... }
  public String toUniversalString() { ... }
  public String toString()   { ... }
}
```


Time1Test.java

1.Time1 class Instance Variables & Methods

- The instance variables hour, minute and second are each declared **private**
  - private instance members are not accessible outside the class.

```
Time1 time = new Time1();
System.out.println( time.toString() );
```

```
The initial standard time is: 12:00:00 AM
```

```
System.out.println( time.hour );
```

%02d:%02d:%02d

```
The initial universal time is: 00:00:00
The initial standard time is: 12:00:00 AM
```

- Instance Methods:
  - **toUniversalString and toString**

```
System.out.println( time.toUniversalString() );
System.out.println( time.toString() );
```

Complete the toUniversalString method

```
public String toString() {
      return String.format( "%d:%02d:%02d %s",
         ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 ),
         minute, second, ( hour < 12 ? "AM" : "PM" ) );
 }
```


1.Time1 class Instance Variables & Methods

DEMO

- Method setTime declares **three** int parameters and uses them to set the time.
  - test each argument to determine whether the value is outside the proper range.
    - If it is out of range, set the value to zero

```
time.setTime( 13, 27, 6 );
```

```
Universal time after setTime is: 13:27:06
```

```
time.setTime( 99, 99, 99 );
```

```
Universal time: 00:00:00
```

```
public void setTime( int h, int m, int s ) {
     hour = = ( ( h >= 0 && h < 24 ) ? ……..
   }
```

Complete the setTime method

2.Case Study: Time2

```
public class Time2 {
  private int hour;   // 0 - 23
  private int minute; // 0 - 59
  private int second; // 0 – 59
 ...
```

- Note:
  - No constructor has been defined in Time1 class. We can only use the default one

Time2.java

- Case Study: Time2 class
  - Add 5 overloaded constructors
    - Overloaded constructors enable objects of a class to be initialized in different ways.
    - To overload constructors, simply provide multiple constructor declarations with different signatures.
      - Recall that the compiler differentiates signatures by the number of parameters, the types of the parameters and the order of the parameter types in each signature.
  - Add getHour, getMinute, getSecond methods
  - Add setHour, setMinute, setSecond methods
  - Modify the toString() and toUniversalString() methods

2.Case Study: Time2 Overloaded Constructors

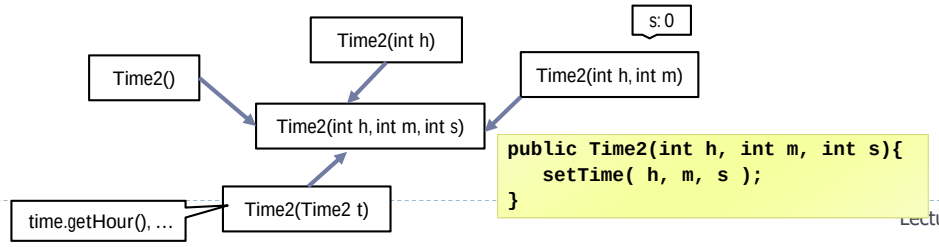
Time2Test.java

- Five overloaded constructors that provide convenient ways to initialize objects.
  - The compiler **invokes** the **appropriate** constructor by matching the **number**, **types** and **order of the types** of the arguments specified in the constructor call with the number, types and order of the types of the parameters specified in each constructor declaration.

```
Time2 t1 = new Time2();             // 00:00:00
Time2 t2 = new Time2( 2 );          // 02:00:00
Time2 t3 = new Time2( 21, 34 );     // 21:34:00
Time2 t4 = new Time2( 12, 25, 42 ); // 12:25:42
Time2 t5 = new Time2( 27, 74, 99 ); // 00:00:00
Time2 t6 = new Time2( t4 );         // 12:25:42
```


2.Case Study: Time2 Overloaded Constructors

- Such a constructor simply initializes the object as specified in the constructor's body
- Using this in method-call syntax as the first statement in a constructor's body **invokes another constructor** of the same class.
- Popular way to reuse initialization code provided by another of the class's constructors rather than defining similar code in the no-argument constructor's body.
- Once you declare any constructors in a class, the compiler will not provide a default constructor.
- Standard constructor: Time2(int h, int m, int s)




```
public Time2(int h, int m, int s){
   setTime( h, m, s );
}
```


Exercise 1: Complete all constructors

DEMO

```
public class Time2 {
  private int hour;   // 0 - 23
  private int minute; // 0 - 59
  private int second; // 0 – 59
  public void setTime( int h, int m, int s ) {
      setHour( h );   // set the hour
      setMinute( m ); // set the minute
      setSecond( s ); // set the second
  }
  public void setHour( int h ){
      hour = ( ( h >= 0 && h < 24 ) ? h : 0 );
  }
...
```


2.Case Study: Time2 Get & Set methods

Kahoot!

- It would seem that providing set and get capabilities is essentially the same as making a class's instance variables **public**.
  - A public instance variable can be read or written by any method that has a reference to an object that contains that variable.
  - If an instance variable is declared **private**, a public get method certainly allows other methods to access it, but the get method can **control** how the client can **access** it.
  - A public set method can—and should—carefully scrutinize attempts to modify the variable's value to ensure **valid** values.
    - We can check and only modify if the parameter is a valid value
- Although set and get methods provide access to private data, it is **restricted** by the implementation of the methods

`time.hour`

Advantages

```
public void setHour( int h ){
      hour = ( ( h >= 0 && h < 24 ) ? h : 0 );
  }
```

Validation

3.Composition Date

- Class Date
  - Instance variables: day, month and year to represent a date
  - The constructor receives three int parameters. It also validate day if it's out of range or invalid
  - The toString method return the object's string representation.

```
class Date {
   private int month; // 1-12
   private int day;   // 1-31 based on month
   private int year;  // any year
   public Date( int theMonth, int theDay, int theYear ) {
      month = checkMonth( theMonth ); // validate month
      year = theYear; // could validate year
      day = checkDay( theDay ); // validate day
   ...
   public String toString() {
      return String.format( "%d/%d/%d", month, day, year );
   }
...
```

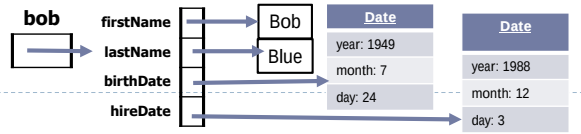

EmployeeTest.java

3.Composition

- A class can have references to objects of other classes as members.
- This is called composition and is sometimes referred to as a **has-a relationship**.
- Example: Employee and hire date (Date)
- Class Employee
  - Instance variables: firstName, lastName, birthDate and hireDate
  - Members firstName and lastName are references to String objects
  - Members birthDate and hireDate are references to Date objects

```
Date birth = new Date(7, 24, 1949);
Date hire = new Date(3,12,1988);
Employee bob = new Employee("Bob", "Blue", birth, hire);
System.out.println(bob);
```

```
Blue, Bob; Hired: 3/12/1988 Birthday: 7/24/1949
```



3.Composition Employee

- Employee & Date : has-a relationship
  - Employee Class
  - Date class

```
class Date {
  private int month; // 1-12
  private int day;   // 1-31 based on month
  private int year;
  public Date( int theMonth, int theDay,
    int theYear ){
  ...
}
```

```
public class Employee {
   private String firstName;
   private String lastName;
   private Date birthDate;
   private Date hireDate;
  ...
```

```
public Employee(String first, String last,
  Date birth, Date hire ) {
    firstName = first;
    lastName = last;
    this.birthDate = birth;
    hireDate = hire;
}
```


3.Composition Java Source Files

- When you compile a .java file containing more than one class, the compiler produces a **separate class** file with the .class extension for every compiled class.
- When one source-code (.java) file contains multiple class declarations, the compiler places both class files for those classes in the same directory.
- A source-code file can contain only **one public class**—otherwise, a compilation error occurs.

EmployeeTest.java

```
class Date {
...
}
class Employee {
   ...
}
public class EmployeeTest {
  public static void main( String args[] ) {
 ...
}
```

Exercise 2

CodeRunner

- A 3x3 matrix is represented by the following array:
  - { {29, 28, 27}, {16, 15, 14}, {3, 2, 1} };

```
class MyMatrix {
  private int[][] data;
  private final static int  SIZE=3;
  public MyMatrix() {  //complete this }
  public MyMatrix(int x11, int x12, int x13, int x21, int x22, ...
```

```
public String toString() {
    StringBuffer sb = new StringBuffer("");


     return sb.toString();
}
```

```
MyMatrix myMatrix2  = new MyMatrix(1, 2, 3, 4, 5, 6, 7, 8, 9);
System.out.println(myMatrix2 );
```

```
[1, 2, 3]
[4, 5, 6]
[7, 8, 9]
```


Exercise 3

CodeRunner

- Complete the add method. This method takes two MyMatrix as parameters, adds and returns a new MyMatrix object.

```
public MyMatrix add(MyMatrix other) {
  MyMatrix result = new MyMatrix();




  return result;
}
```

```
[30, 30, 30]
[20, 20, 20]
[10, 10, 10]
```

```
MyMatrix myMatrix2  = new MyMatrix(1, 2, 3, 4, 5, 6, 7, 8, 9);
MyMatrix myMatrix1  = new MyMatrix(29, 28, 27, 16, 15, 14, 3, 2, 1);
MyMatrix result = myMatrix1.add(myMatrix2);
System.out.println(result);
```