Chap11Yourname.java

Classes

```
/*
 * Chap11 Classes
 * Time.java
 */

import java.io.PrintStream;

public class Chap11 {

	public static void main(String[] args) {
		PrintStream out = System.out;
```

//Defining a class creates a new object type with the same name.
//Every object belongs is an instance of some class.

//A class definition is like a template for objects: it specifies what
//attributes the objects have and what methods can operate on them.

//The methods that operate on an object type are defined in the class for
//that object.

11.1 - 11.3 Instance variables, constructors

```
out.println("11.1 - 11.3 Instance variables, constructors");
//Now Create the Time class: Time.java
```

11.1 - 11.3 Instance variables, constructors

```
/*
 * Time class of Chap11.java
 */

//One common reason to define a class is to encapsulate related data in
//an object that can be treated as a single unit. (Data encapsulation)

public class Time {
//The Time class is public which means it can be used in other classes.

//Attributes are also called instance variables. What instance variables do we need?
```

11.1 - 11.3 Instance variables, constructors

```
private int hour;
private int minute;
private double second;
//The instance variables are private, which means they can only be accessed
//from inside the Time class.
//Private instance variables help keep classes isolated from each other so that
//changes in one class will not require changes in other classes.
//(information hiding).
```

11.1 - 11.3 Instance variables, constructors

```
//After declaring the instance variables, the next step is to define a
//constructor, which is a special method that initializes the instance variables.
public Time() {
//The name of the constructor is the same as the name of the class.
//Constructors have no return type or return value.
    this.hour = 0;
    this.minute = 0;
    this.second = 0;
    //This constructor does not take any arguments.
```

11.1 - 11.3 Instance variables, constructors

```
        //The name this is a keyword that refers to the object we are creating.
        //Each line initializes an instance variable to 0.

        //This constructor make it possible to create a Time object with default
        //attributes.

        //Now create a Time object, time1, in Chap11.java.
    }
```

11.1 - 11.3 Instance variables, constructors

```
//Create a Time object, time1.
Time time1 = new Time();
//When we invoke new, Java creates the object and calls the constructor to
//initialize the instance variables. Then new returns a reference to the new
//object.

//Now create a value constructor in the Time class.
```

11.1 - 11.3 Instance variables, constructors

```
//Now create a value constructor.
public Time(int hour, int minute, double second) {
    this.hour = hour;
    this.minute = minute;
    this.second = second;
```

11.1 - 11.3 Instance variables, constructors

```
        if (this.second >= 60) {
            this.second -= 60;
            this.minute += 1;
        }
        if (this.minute >= 60) {
            this.minute -= 60;
            this.hour += 1;
        }
        if (this.hour >= 24) {
            this.hour -= 24;
        }
    }
```

11.1 - 11.3 Instance variables, constructors

```
//To invoke this value constructor, we have to provide arguments after the new
//operator.

//The proceeding constructors are overloading. Overloading constructors provide
//the flexibility to create an object first and then fill in the attributes.
//Now create a Time object, time2, in Chap11.java with
//hour = 24, minute = 59 and second = 60
```

11.1 - 11.3 Instance variables, constructors

```
//Create a Time object, time2, in Chap11.java with
//hour = 24, minute = 59 and second = 60.
Time time2 = new Time(24, 59, 60);
out.println();
```

11.5 Displaying objects

```
out.println("11.5 Displaying objects");
out.println(time1);
out.println(time2);
//The results are not what we want.
//Create an instance method printTime()
```

11.5 Displaying objects

```
//Create an instance method printTime()
public void printTime() {
    System.out.printf("%02d:%02d:%04.1f\n", this.hour, this.minute, this.second);
}
//An in stance method is not static. We have to invoke it on an instance of the
//class.
```

11.5 Displaying objects

```
time1.printTime();
time2.printTime();
out.println();
```

11.4 Getters and setters

```
out.println("11.4 Getters and setters");
out.println(time1.hour);
//Because the instance variables of Time are private. We cannot access them
//from outside the Time class.

//To access the instance variables from outside, we need to create methods
//in the Time class called getters.
//To modify the instance variables from outside, we need to create methods
//in the Time class called setters.
```

11.4 Getters and setters

```
out.println("11.4 Getters and setters");
//out.println(time1.hour);
//Because the instance variables of Time are private. We cannot access them
//from outside the Time class.

//To access the instance variables from outside, we need to create methods
//in the Time class called getters.
//To modify the instance variables from outside, we need to create methods
//in the Time class called setters.
```

11.4 Getters and setters

```
//Create getters and setters.
public int getHour() {
    return this.hour;
}

public int getMinute() {
    return this.minute;
}

public double getSecond() {
    return this.second;
}
//These methods are instance methods.
```

11.4 Getters and setters

```
public void setHour(int hour) {

}

public void setMinute(int minute) {

}

public void setSecond(double second) {

}
//Recall that Strings are immutable. To make a class immutable we should not
//provide setters.
```

11.4 Getters and setters

```
public void setHour(int hour) {
    this.hour = hour;
}

public void setMinute(int minute) {
    this.minute = minute;
}

public void setSecond(double second) {
    this.second = second;
}
//Recall that Strings are immutable. To make a class immutable we should not
//provide setters.
```

11.4 Getters and setters

```
out.println(time1.getHour());
out.println(time2.getHour());
time2.setHour(2);
time1.printTime();
time2.printTime();
out.println();
```

11.6 The toString method

```
out.println("11.6 The toString method");
String name = "John von Neumann";
out.println(name);
//Every object type has a toString method that returns a String
//representation of the object. Create a toString method for Time class.
out.println(time1);
out.println(time2);
out.println();
```

11.6 The toString method

```
public String toString() {
    return String.format("%02d:%02d:%04.1f",
            this.hour, this.minute, this.second);
}
```

11.7 The equals method

```
out.println("11.7 The equals method");
//The == operator checks whether variables refer to the same object.
//Write a equals method for Time class checking whether variables have
//the same value.
out.println(time1.equals(time2));
out.println();
```

11.7 The equals method

```
//Is it a static or instance method? What is the return type?
```

11.7 The equals method

```
//Is it a static or instance method? What is the return type?
public boolean equals(Time that) {



}
```

11.7 The equals method

```
//Is it a static or instance method? What is the return type?
public boolean equals(Time that) {
      return this.hour == that.hour
                  && this.minute == that.minute
                  && this.second == that.second;
}
```

11.8 Pure methods

```
out.println("11.8 Pure methods");
Time goToBed = new Time(23, 0, 0);
//If you wake up 10 hours, 20 minutes and 30 seconds later, what will be
//the time?
Time sleep = new Time(10, 20, 30);
//Write a method add(Time t1, Time t2) and invoke
//Time.add(goTOBed, sleep) to solve the problem. Is it a static or
//instance method?
out.println(Time.add(goToBed, sleep));
```

11.8 Pure methods

```
public static Time add(Time t1, Time t2) {


}
```

11.8 Pure methods

```
public static Time add(Time t1, Time t2) {
        return new Time(t1.hour + t2.hour, t1.minute + t2.minute,
                        t1.second + t2.second);
}
```

11.8 Pure methods

```
//Write an method add(Time t) and invoke goToBed.add(sleep) to solve the
//problem. Is it a static or instance method?
out.println(goToBed.add(sleep));
```

11.8 Pure methods

```
public Time add(Time t) {


}
```

11.8 Pure methods

```
public Time add(Time t) {
    return new Time(this.hour + t.hour,  this.minute + t.minute,
                this.second + t.second);
}
```

11.8 Pure methods

```
//Write a method modifier(Time t) that does not return a new Time object
//but modifies the existing Time object. Is it a static or instance method?
out.println(goToBed.modifier(sleep));
out.println(goToBed);
//The method modifier(Time t)has modified the arguments of goToBed.
```

11.8 Pure methods

```
public Time modifier(Time t1) {
```

11.8 Pure methods

```
public Time modifier(Time t1) {
    this.second += t1.second;
    this.minute += t1.minute;
    this.hour += t1.hour;
```

11.8 Pure methods

```
        if (this.second >= 60) {
            this.second -= 60;
            this.minute += 1;
        }
        if (this.minute >= 60) {
            this.minute -= 60;
            this.hour += 1;
        }
        if (this.hour >= 24) {
            this.hour -= 24;
        }

        return this;
    }
```

11.8 Pure methods

```
//Methods like add are pure methods:
//(1) The return value of the pure methods solely depends on its
//                arguments. Hence, if you invoke the pure methods with the same set
//                of arguments, you will always get the same return values.
//(2) They do not have any side effects like output to I/O devices.
//                In computer science, a function or expression is said to have a
//                side effect if it modifies some state outside its scope or has
//                an observable interaction with its calling functions or the outside
//                world besides returning a value.
```

11.8 Pure methods

//(3) They do not modify the arguments which are passed to them.

//Math.random() is impure because it may return different values
//out.println() is impure because it causes output to an I/O device as a
//side effect.
//goToBed.modifier(Time t) is impure because it modifies the arguments.

//To make a class immutable, we should not provide modifier methods.
out.println();