

Lab #10

Objects and Classes

Case Study: A "Time" class

- Suppose we want to be able to work with values representing clock times to 1 minute precision.
 - What information do we have to store to represent a time? How do we store that information?
 - What operations might we want to do with a Time value?

What to store in "Time"?

- Two integers:
 - **hour**: the hour number ($0 \leq \text{hour} \leq 23$)
 - **minute**: the minute number ($0 \leq \text{minute} \leq 59$)

```
class Time
{
    public int hour;
    public int minute;
}
```

- Alternatives?

Declaring and creating Time objects

```
// DECLARE VARIABLES
```

```
Time time1; // time is null here
```

```
...
```

```
// USING Time OBJECT
```

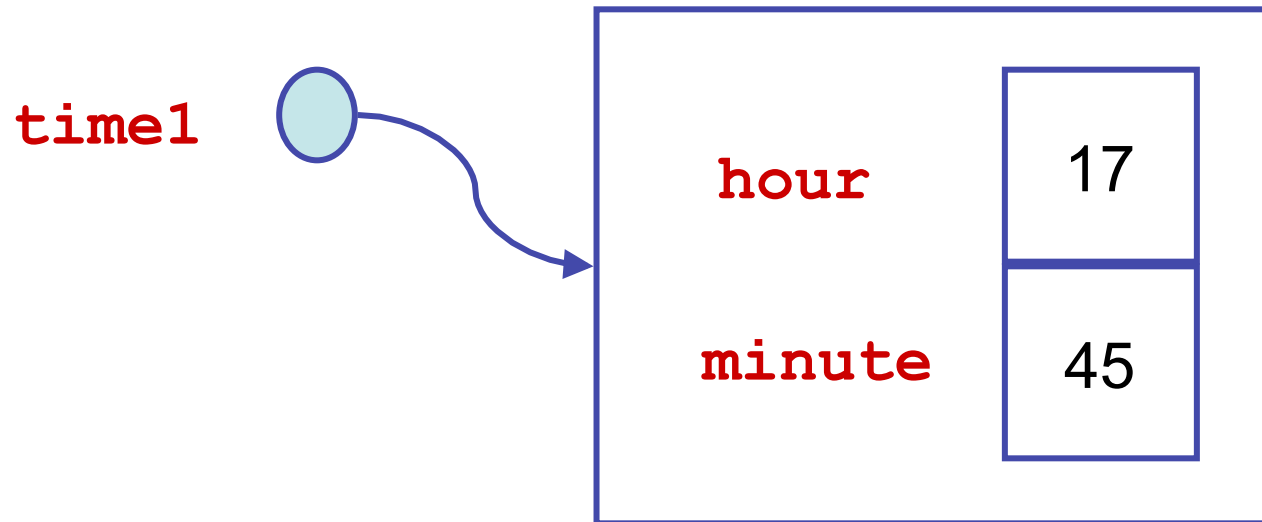
```
...
```

```
time1 = new Time( ); // We now have a Time object  
// but it contains ? for values
```

```
time1.hour = 17;
```

```
time1.minute = 45; // time1 now represents 17:45
```

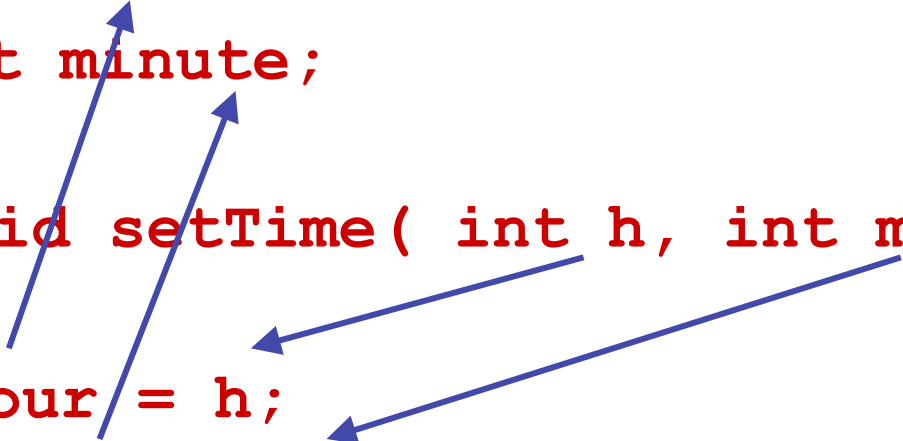
What do we have?



A method to set the time

```
class Time
{
    public int hour;
    public int minute;

    public void setTime( int h, int m )
    {
        this.hour = h;
        this.minute = m;
    }
}
```



The diagram illustrates the parameter passing in the `setTime` method. It features four blue arrows: one from the parameter `h` to the `hour` attribute, one from the parameter `m` to the `minute` attribute, and two from the `h` and `m` parameters to the assignment expressions `this.hour = h;` and `this.minute = m;` respectively, showing how the method arguments are used to update the object's state.

Usage

```
// DECLARE VARIABLES
```

```
Time time1;
```

```
// time is null here
```

```
...
```

```
// USING Time OBJECT
```

```
...
```

```
time1 = new Time( );
```

```
time1.setTime( 17, 45 );
```

```
// time1 now represents 17:45
```

This method is different!

- Did anyone see what was “missing” from the method?

```
public void setTime( int h, int m )  
{  
    this.hour = h;  
    this.minute = m;  
}
```

- The word **static** does not appear in the method header.

Instance methods

- When the word **static** does not appear in the method header, this means that the method can be called via a variable declared to be of a type matching the class name. This is called an "instance" method.
- An instance method can make use of the variables defined in the class.
- The result: the method will produce different results for different object instances.

Instance methods

- For example,

```
Time time1;
```

```
Time time2;
```

```
time1 = new Time( );
```

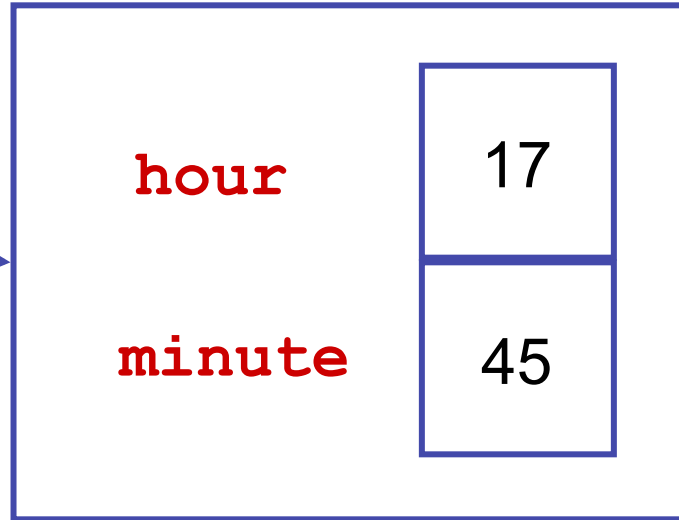
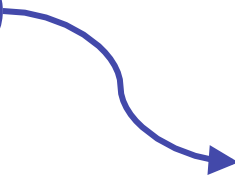
```
time2 = new Time( )
```

```
time1.setTime( 17, 45 ); // time1 is 17:45
```

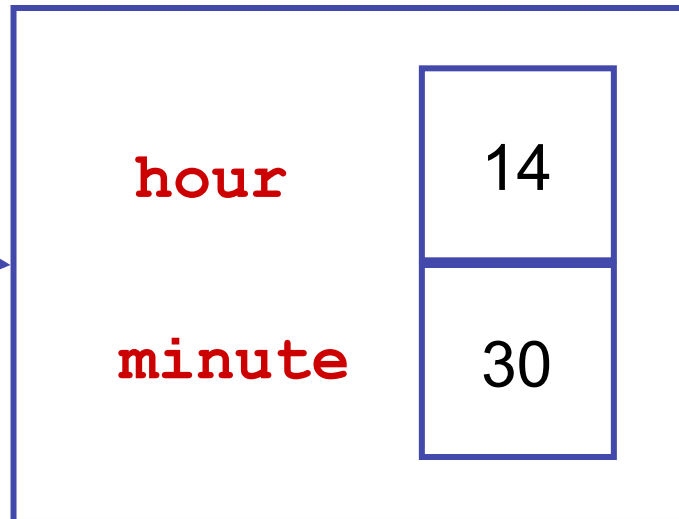
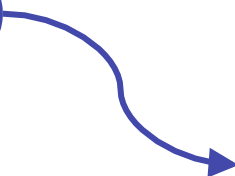
```
time2.setTime( 14, 30 ); // time2 is 14:30
```

What do we have?

time1



time2



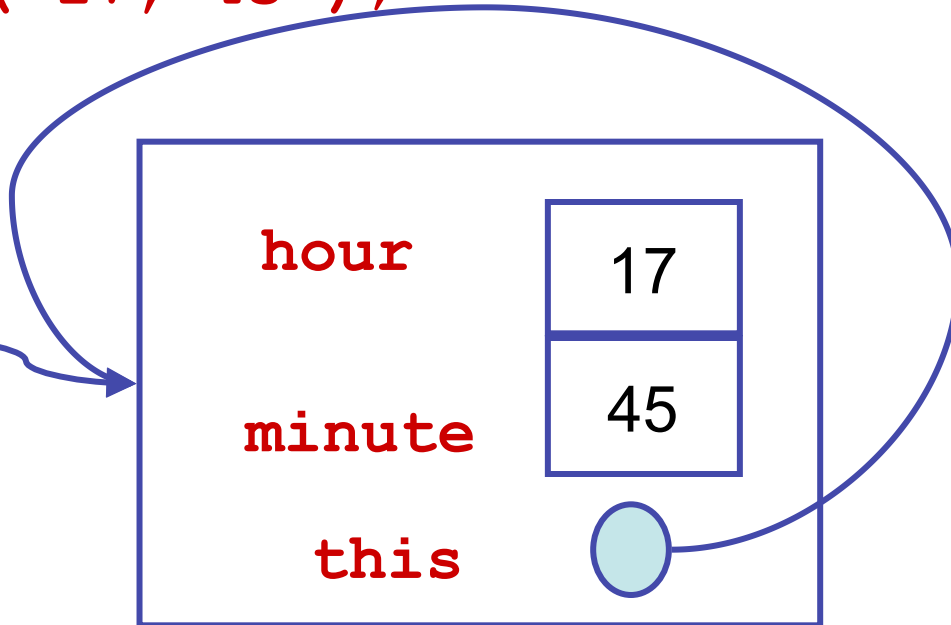
this

- Objects `time1` and `time2` use the same code in class `Time` to set their `own` copy of `hour` and `minute`.
- When we want to refer to "the object on which I was called", we use `this`.

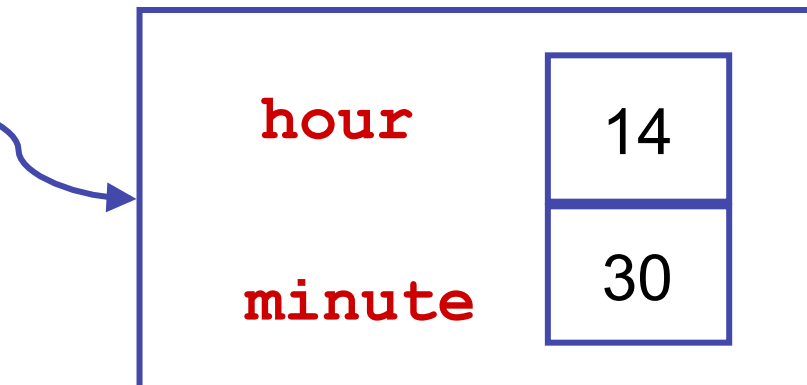
this

time1.setTime(17, 45);

time1



time2



Information Hiding

- If we want to ensure that:
 - **hour**: must be in range $0 \leq \text{hour} \leq 23$
 - **minute**: must be in range $0 \leq \text{minute} \leq 59$then direct access to these variables should not be permitted.

```
class Time
{
    private int hour;
    private int minute;
}
```

- Instead, access should be provided through **setTime**, and we can adjust the values if needed.

Revised version of setTime

```
public void setTime( int h, int m )
{
    // If minutes value is too large, adjust it
    // by mod 60, and add to hours value.

    if ( m > 59 )
    {
        h = h + m / 60;    // determine hours to add
        m = m % 60;        // puts minutes in range
    }
    else
    {
        ; // do nothing
    }
    this.hour = h % 24;    // puts hours in range
    this.minute = m;
}
```

Accessors

- With the variables now declared to be **private**, we need to provide a way for other classes to ask for the values.

```
public int getHours()  
{  
    return hour;  
}  
public int getMinute( )  
{  
    return minute;  
}
```


Compare times for equality

- Suppose we want a method that checks whether one time object is equal to another.

- One approach: a **static** method

```
public static boolean isEqual( Time t1, Time t2 )
```

This would be called as **Time.isEqual(t1, t2)**

- Alternative: an instance method

```
public boolean isEqual( Time t2 )
```

This would be called as **t1.isEqual(t2)**

The **static** method

```
public static boolean isEqual( Time t1, Time t2 )  
{  
    return (t1.hour == t2.hour) &&  
           (t1.minute == t2.minute);  
}
```

- If the method is inside the class **Time**, it can access the private variables inside the class.

The instance method

```
public boolean isEqual( Time t2 )  
{  
    return (this.hour == t2.hour) &&  
           (this.minute == t2.minute);  
}
```

- In this case, we are comparing “ourself” to another **Time** value.

Exercise 1: Add two *instance* methods to provided Time class

- `public boolean isBefore(Time t)`

that returns true is the Time `t` is before the current time object

- `public Time duration(Time t)`

This method calculates the time duration from the current time to the given time `t`. Return the duration as a Time object.

CONTINUE ON THE NEXT PAGE

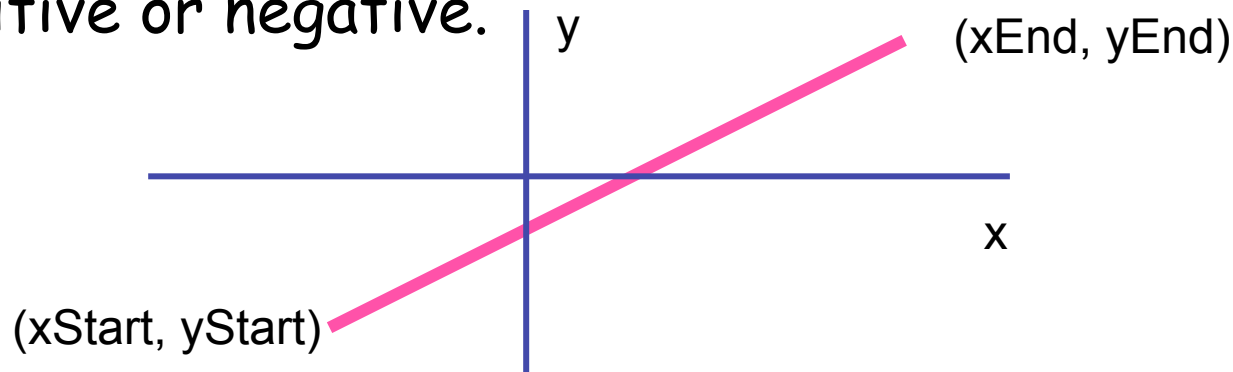
Exercise 1: Add two **instance** methods to provided Time class

Also test your two methods using Junit tests from the provided file `TimeTest.java`. You will need to use the Test menu button in DrJava instead of the Run button to execute the tests.

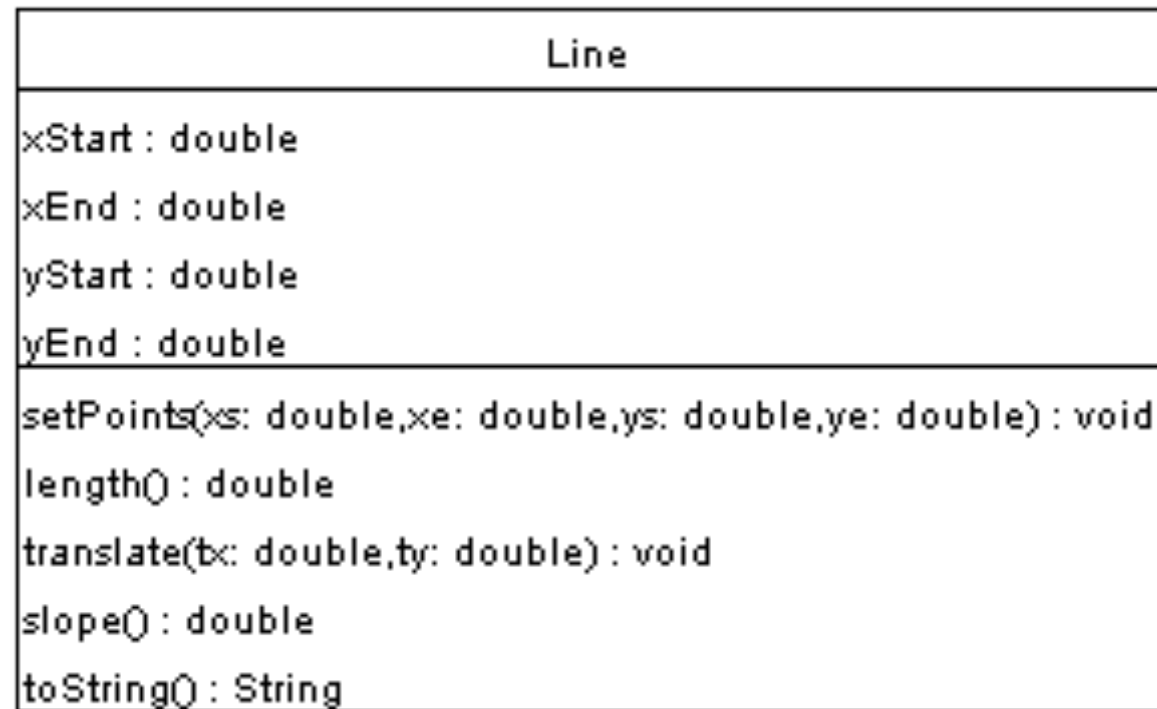
- To do Junit testing, load your `Time.java` solution and the provided `TimeTest.java` to DrJava. Compile both files, and then press “Test” menu button in DrJava. `TimeTest.java` contains many tests. Those tests for which your solution computes the correct answer will be displayed in green and those for which your solution fails will be displayed in red. Make sure all the provided tests are displayed in green i.e. that they pass.
- You will have to do similar testing in your Assignment 5

A "Line" segment class

- Design a Java class called **Line** that will store information about a line segment, where the line segment is specified by the coordinates, (x,y) , of its two endpoints. Provide operations to work with segments.
- Each Line object starts at a point $(xStart, yStart)$ and ends at a point $(xEnd, yEnd)$, where $xStart$, $yStart$, $xEnd$, and $yEnd$ are real-valued numbers that could be positive or negative.



UML diagram for **Line**



If the method is **not underlined** that it is an **instance method**.
Underlined methods are **static** methods.

Thus, this UML diagram tells us that all the methods in this class are instance methods

Method descriptions

- **setPoints method**: Set the start and end points of the line segment.
 - Name of method: **setPoints(...)**
 - Parameters to the method: **xs, ys, xe, ye**
 - Returns: (nothing)
 - This method modifies the Line object

Method descriptions

- **length method**: Returns the length of the line segment
 - Name of method: **length()**
 - Input parameters to the method: none.
 - Method returns: length of the line segment (a real value)

$$\sqrt{(y_e - y_s)^2 + (x_e - x_s)^2}$$

Method descriptions

- **slope method:** Returns the slope of the line segment.
 - The slope is $(y_e - y_s) / (x_e - x_s)$ (watch out for vertical lines - their slope is infinite!)
 - Name of method: **slope ()**
 - Parameters to the method: none.
 - The methods returns: the slope of the line segment (a real value)

Method descriptions

- **translate method**: Translate the segment by (tx, ty) , where tx and ty are any positive or negative real values.
 - A **translation** of a line segment represents "sliding" the entire segment. The value tx is added to the x coordinates of the start and end of the segment, and the value ty is added to the y coordinates of the start and end of the segment.
 - Name of method: **translate(...)**
 - Parameters to the method: **tx, ty**
 - Results: (none)
 - This method modifies the Line object

Method descriptions

- **toString method**: Returns a **String** with information about the line segment.
 - The string that is returned for a line segment with (for example) start point (0.0,1.0) and end point (3.5,-1.2) should be:
"Line from (0.0, 1.0) to (3.5, -1.2)"
- Name of method: **toString ()**
- Parameters to the method: (none)
- Results: a **String** in the above format

Exercise 2

- Implement the class **Line** and all of its methods.

Recall, by looking at UML diagram or method descriptions, that all the methods of the Line class are supposed to be **instance methods**.

- Test the class using the main method in the provided file **LineTest.java**

Extra Exercise 3

We want to realize a program for the administration of a parking lot formed by a collection of parking places. For each parking place the following information has to be stored:

- Whether it is free or occupied;
- Licence plate of the car, when occupied (a string);
- Time since when it is occupied (two integers, for hours and minutes).

Write a class `ParkingPlace`, for handling a parking place, that implements the following methods:

- `ParkingPlace()`: constructor without parameters that constructs a parking place that is initially free;
- `String toString()`: that returns "-----" if the parking place is free, and the licence plate of the car, if the parking place is occupied;
- `void carArrives(String plate, int hour, int minutes)`: modifies the state of the parking place by setting it to occupied, sets the plate of the car that occupies the place to plate, and sets the time since when it is occupied to hour and minutes; if the parking place is already occupied, the method does nothing;
- `void carLeaves()`: modifies the state of the parking place by setting it to free;
- `boolean free()`: returns true if the parking place is free, false otherwise;
- `String getCar()`: returns the plate of the car that occupies the parking place, if the place is occupied, null otherwise;
- `int getHour()`: returns the hour since when the parking place is occupied, if the place is occupied, -1 otherwise;
- `int getMinutes()`: returns the minutes since when the parking place is occupied, if the place is occupied, -1 otherwise;

Exercise 3 Testing

Use the example program [TestParkingPlace.Java](#) to test the class you have developed.