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()   { ... }
}
```

1.Time1 class
Instance Variables & Methods

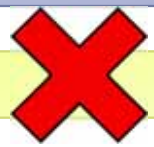
Time1Test.java

- 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

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

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

```
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

- 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

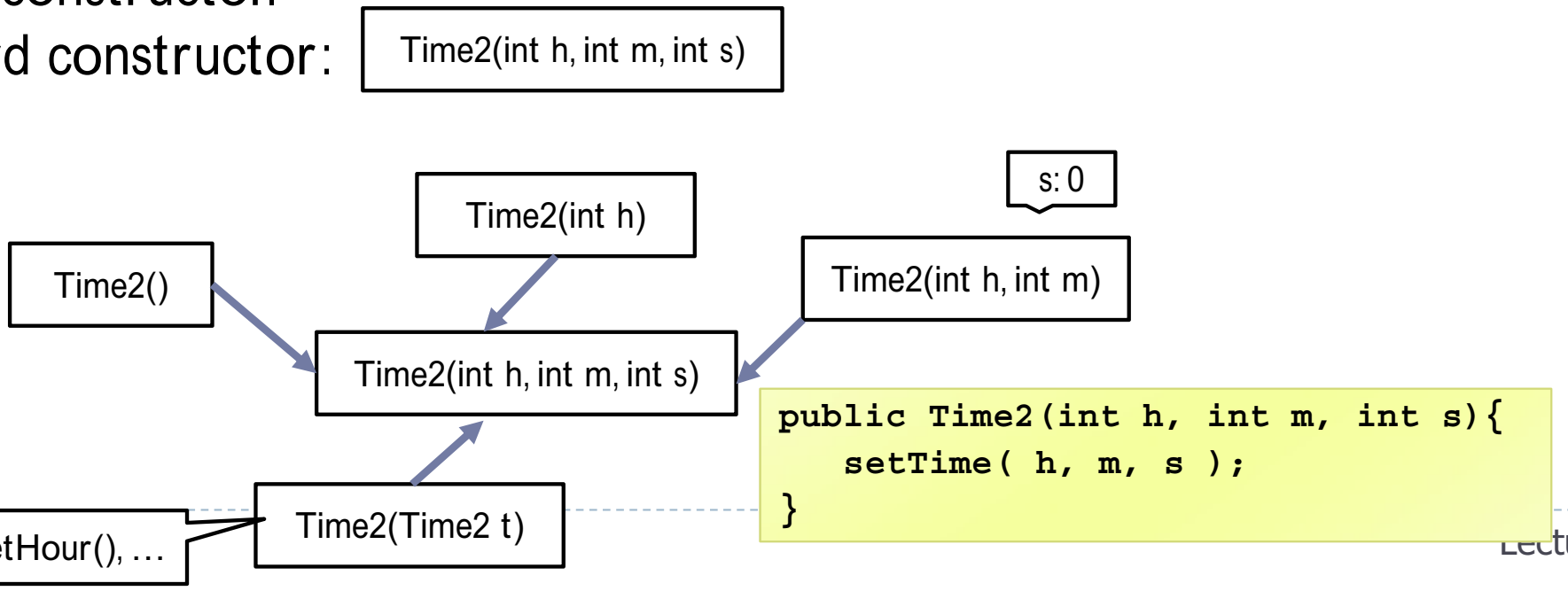
Time2Test.java

- 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:



Exercise 1: Complete all constructors


```
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

- 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 );
   }
...
```