```
#-------------------------------------------------------------------------
# File:         Overview.txt
# Date:         Wed Oct 12 18:34:35 EDT 2011
# Author(s):    Ellery Coleman <ellerycoleman@fas.harvard.edu>
# Abstract:     Overview of HW1 Solution.
#-------------------------------------------------------------------------
# Revision: $Id$
#-------------------------------------------------------------------------


My solution to homework one is contained in a single file; Elevator.java.
This file implements the Elevator class according to the HW1 Spec.

My Elevator class uses mnemonic constants for static values.  These include
baseFloor, maxFloor, maxCapacity, UP, and DOWN.  I keep track of destination
requests in an array called destRequests, and I also keep track of how many
passengers are destined to a particular floor with an array called
passengersToFloor.  Both of these arrays are managed by calls to
boardPassenger() and stop().

Based on the language of the HW1 Spec, my elevator sweeps the building
continuously until you hit CTRL+C.


The solution is compiled into a jar file; hw1.jar.
Use the following command to run the solution from the command line:

$ java -jar hw1.jar


Again, to stop the elevator from running enter CTRL+C.
All of the source code, documentation, and javadocs for this solution are
contained in the hw1.jar file.  You can access them as follows:


$ jar xf hw1.jar
$ cat Overview.txt
$ cd javadoc
```

```
/*--------------------------------------------------------------------------
# File:         Elevator.java
# Date:         Tue Sep 13 17:13:24 EDT 2011
# Author:       Ellery Coleman <ellerycoleman@fas.harvard.edu>
# Abstract:     Implements an Elevator class for cscie160, hw1.
#--------------------------------------------------------------------------
# Revision: $Id: Elevator.java 4 2011-10-12 22:08:43Z ellery $
#--------------------------------------------------------------------------*/
package cscie160.hw1;




public class Elevator
{

    //-----------------------
    //     Constants
    //-----------------------


   /**
    * defines the base floor in the building
    */
    public static final int baseFloor   = 1;



   /**
    * defines the top floor in the building
    */
    public static final int maxFloor    = 7;


   /**
    * defines the max capacity of an elevator
    */
    public static final int maxCapacity = 10;


   /**
    * The direction of an elevator is set with an
    * integer variable; 1 for UP.
    */
    public static final int UP          = 1;


   /**
    * The direction of an elevator is set with an
    * integer variable; 0 for DOWN.
    */
    public static final int DOWN        = 0;




    //-----------------------
    //     Constructor
    //-----------------------

   /**
    * A constructor that doesn't take any arguments; initializes
    * the elevator with zero passengers, first floor position, and
    * a direction of UP.
    */
    public Elevator()
    {   System.out.println("Initializing elevator...\n");
        this.passengers=0;
        this.floor= 1;
        this.direction= UP;
    }




    //-----------------------
    //     Data Members
    //-----------------------
    private int floor;
    private int passengers;
    private int direction; // 1 is up, 0 is down
```

```
   private int destRequests[]= new int[maxFloor+1];
   private int passengersToFloor[]= new int[maxFloor+1];




   //-----------------------
   //     Method Members
   //-----------------------

   /*-------------------------------------------------------------------------
   | method name: toString
   | return type: String
   |    Abstract: Overrides toString() method of java.lang.Object;
   |              returns string with status of elevator.
   +-------------------------------------------------------------------------*/
  /**
   *  Returns string with status of elevator.
   */
   public String toString()
   {   String status,requests;
       requests="";
       status= "+--------Elevator-----------"               + "\n" +
               "|         current Floor: "  + this.floor       + "\n" +
               "|    current passengers: "  + this.passengers  + "\n" +
               "|     current direction: "  +
                  ((this.direction == UP) ? "up":"down")   + "\n" +
               "|  destination requests: ";

       for(int i=1; i<=maxFloor; i++)
       {   if((destRequests[i] != 0)  || (passengersToFloor[i] != 0))
           {   requests+= "\n" +
                          "|      Floor_" + (i) + "--> requests: " +
                          destRequests[i] + ", " +
                          " passengers destined for floor: " +
                          passengersToFloor[i] + "\n";
           }
       }
       if(requests.isEmpty())
       {   status+= "none\n";
       }
       else
       {   status+= requests;
       }

       status+= "+---------------------------\n\n\n\n";
       return status;
   }




   /*-------------------------------------------------------------------------
   | method name: move
   | return type: void
   |    Abstract: Moves elevator by one floor depending on current
   |              direction.
   +-------------------------------------------------------------------------*/
  /**
   * Moves elevator by one floor depending on current direction.
   * Changes direction as appropriate.
   */
   public void move()
   {   if((this.direction == UP)  && (this.floor < maxFloor))
       {   this.floor++;
       }
       else if((this.direction == UP)  && (this.floor == maxFloor))
       {   this.direction=DOWN;
           this.floor--;
       }
       else if ((this.direction == DOWN)  && (this.floor > baseFloor))
       {   this.floor--;
       }
       else if ((this.direction == DOWN)  && (this.floor == baseFloor))
       {   this.direction=UP;
           this.floor++;
       }

       if(destRequests[this.floor] > 0)
       {   stop();
```

```
        }
    }



    /*--------------------------------------------------------------------
    | method name: stop
    | return type: void
    |    Abstract: Stops the elevator, does the appropriate book keeping,
    |              and then displays the state of the elevator after the
    |              processing.
    +-------------------------------------------------------------------*/
   /**
    * Stops the elevator, does the appropriate book keeping,
    * and then displays the state of the elevator after the
    * processing.
    */
    public void stop()
    {  System.out.print("\n\n\nElevator stopped at floor " + this.floor + ", ");
       int unloading= destRequests[floor];
       destRequests[floor]=0;
       passengersToFloor[floor]-= unloading;
       passengers-= unloading;
       System.out.println("dropped off " + unloading + " passenger(s).");
       System.out.println(this.toString());
    }







    /*--------------------------------------------------------------------
    | method name: boardPassenger
    | return type: void
    |    Abstract: Adds a passenger to the elevator and handles the
    |              appropriate accounting.
    +-------------------------------------------------------------------*/
   /**
    * Adds a passenger to the elevator and handles the appropriate class
    * book keeping of increasing the passenger count on the elevator,
    * registering the destination request, and increasing the count of
    * passengers headed to the destination floor.
    */
    public void boardPassenger(int floor)
    {   System.out.println("Boarding one passenger for floor " + floor + ".");
        this.passengers++;
        destRequests[floor]++;
        passengersToFloor[floor]++;
    }







    //-----------------------------------
    //     Main Method (test harness)
    //-----------------------------------
   /**
    * A test harness for the Elevator class.  According to HW1 Spec, this
    * method boards 2 passengers for the 2nd floor and 1 for the 3rd floor.
    * In accordance with the Spec the elevator continues to sweep the building
    * after servicing these passengers.
    */
    public static void main(String args[]) throws InterruptedException
    {

        Elevator ev1= new Elevator();
        ev1.boardPassenger(2);
        ev1.boardPassenger(2);
        ev1.boardPassenger(3);

        while(true)
        {   ev1.move();
            Thread.sleep(1000);  //sleep - helps interpret output in real time
        }
    }
}
```

```
[ellery@ssh cs160]$ java -jar hw1.jar
Initializing elevator...

Boarding one passenger for floor 2.
Boarding one passenger for floor 2.
Boarding one passenger for floor 3.



Elevator stopped at floor 2, dropped off 2 passenger(s).
+--------Elevator----------
|         current Floor: 2
|    current passengers: 1
|     current direction: up
|  destination requests:
|      Floor_3--> requests: 1,  passengers destined for floor: 1
+--------------------------









Elevator stopped at floor 3, dropped off 1 passenger(s).
+--------Elevator----------
|         current Floor: 3
|    current passengers: 0
|     current direction: up
|  destination requests: none
+--------------------------



^C[ellery@ssh cs160]$
```

cscie160.hw1

Class Elevator

```
java.lang.Object
  └cscie160.hw1.Elevator
```

```
public class Elevator
extends java.lang.Object
```

Field Summary

static int	**baseFloor** defines the base floor in the building
static int	**DOWN** The direction of an elevator is set with an integer variable; 0 for DOWN.
static int	**maxCapacity** defines the max capacity of an elevator
static int	**maxFloor** defines the top floor in the building
static int	**UP** The direction of an elevator is set with an integer variable; 1 for UP.

Constructor Summary

Elevator() A constructor that doesn't take any arguments; initializes the elevator with zero passengers, first floor position, and a direction of UP.

Method Summary

void	**boardPassenger**(int floor) Adds a passenger to the elevator and handles the appropriate class book keeping of increasing the passenger count on the elevator, registering the destination request, and increasing the count of passengers headed to the destination floor.
static void	**main**(java.lang.String[] args) A test harness for the Elevator class.
void	**move**() Moves elevator by one floor depending on current direction.
void	**stop**() Stops the elevator, does the appropriate book keeping, and then displays the state of the elevator after the processing.

java.lang.String	**toString**() Returns string with status of elevator.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Field Detail

baseFloor

```
public static final int baseFloor
```

defines the base floor in the building

See Also:
Constant Field Values

---

maxFloor

```
public static final int maxFloor
```

defines the top floor in the building

See Also:
Constant Field Values

---

maxCapacity

```
public static final int maxCapacity
```

defines the max capacity of an elevator

See Also:
Constant Field Values

UP

```
public static final int UP
```

The direction of an elevator is set with an integer variable; 1 for UP.

See Also:
Constant Field Values

DOWN

```
public static final int DOWN
```

The direction of an elevator is set with an integer variable; 0 for DOWN.

See Also:
Constant Field Values

Constructor Detail

Elevator

```
public Elevator()
```

A constructor that doesn't take any arguments; initializes the elevator with zero passengers, first floor position, and a direction of UP.

Method Detail

toString

```
public java.lang.String toString()
```

Returns string with status of elevator.

Overrides:
`toString` in class `java.lang.Object`

---

move

```
public void move()
```

Moves elevator by one floor depending on current direction. Changes direction as appropriate.

---

stop

```
public void stop()
```

Stops the elevator, does the appropriate book keeping, and then displays the state of the elevator after the processing.

---

boardPassenger

```
public void boardPassenger(int floor)
```

Adds a passenger to the elevator and handles the appropriate class book keeping of increasing the passenger count on the elevator, registering the destination request, and increasing the count of passengers headed to the destination floor.

---

main

```
public static void main(java.lang.String[] args)
                throws java.lang.InterruptedException
```

A test harness for the Elevator class. According to HW1 Spec, this method boards 2 passengers for the 2nd floor and 1 for the 3rd floor. In accordance with the Spec the elevator continues to sweep the building after servicing these passengers.

Throws:

`java.lang.InterruptedException`
