§ 6.5 CLASSES TIMER AND TIMERTASK

The package *java.util* has two very convenient classes that allow us to schedule a task (a sequence of instructions or a method) that can be performed either (1) at a certain moment of time in the future or (2) repeatedly at regular interval. The names of those classes are *Timer* and *TimerTask*.
Class *Timer* is responsible for actual scheduling, and class *TimerTask* is responsible for the task to be performed.
Class *TimerTask* has a method *run()* that is called repeatedly by the schedule established by class *Timer*.

In order to schedule a certain task, a programmer has to:

1. Type *import java.util.* ;* in the first line of the file;
2. Create an instance of class *Timer* and an instance of class *TimerTask* ;
3. Schedule a task using the methods of the class *Timer schedule(long)* or *scheduleAtFixedRate()*;

An example of the short program that prints out the current time every second is shown below. The program also demonstrates how to use the class Calendar to obtain the current year, month, date, day of week, and time.
Since by the nature of the problem, a scheduled task should be quite short, instead of creating a class that extends the class *TimerTask*, we declare method run() directly in the body of class *TimerTask*.

```
// File Clock.java
import java.util.* ;
public class Clock
{
        public static void main(String[] args)
        {       Timer           tm = new Timer() ;
                long delay  = 3000 ;    //  a delay in milliseconds
                long period = 1000 ;    // a period in milliseconds

                TimerTask       curTime = new TimerTask()
                {       public void run()
                        {       String str = currentDateAndTime() ;
                                System.out.println("Current Time : " + str) ;
                        }
                } ;
                                tm.scheduleAtFixedRate ( curTime, delay, period ) ;
                System.out.println("The timer will start in " + delay/1000.0 + " seconds...") ;

        } //the end of method main()
```

```
    public static String currentDateAndTime()
    {
        String[] day   = {   "Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat" } ;
        String[] month = {   "Jan", "Feb", "Mar", "Apr", "May", "Jun",
                             "Jul", "Aug", "Sep", "Oct", "Nov", "Dec" } ;

        Calendar cl = Calendar.getInstance( TimeZone.getTimeZone("GMT–5:00")) ;
            /*   An object of class Calendar contains all values, in numeric form, relevant to
                 a moment of time when the object was created.
             */
        String  str = "" + day[cl.get(Calendar.DAY_OF_WEEK)] + ", " ;
        str += "" + month[cl.get(Calendar.MONTH)] + " " ;
        str += "" + cl.get(Calendar.DAY_OF_MONTH) + ", " ;
        str += "" + cl.get(Calendar.YEAR) + ", " ;
        str += "" + cl.get(Calendar.HOUR_OF_DAY) + ":" ;
        str += "" + cl.get(Calendar.MINUTE) + ":" ;
        str += "" + cl.get(Calendar.SECOND) ;
        return str ;

    }   // the end of method currentDayAndTime()

}    // the end of class Clock
```

The classes *Timer* and *TimerTask* are very useful for implementing animation because the scheduled tasks are performed by a computer in parallel with the main program and do not interfere with the instructions responsible for user interface (e.g. the methods of *KeyListener* and *MouseListener* interfaces).
The following program moves a circular particle within the program window so that the particle bounces of the walls and let the user to control the speed of motion by using the arrow keys and PgUp / PgDn keys.
As the speed of the particle changes its color changes as well according to the scheme:
$BLACK \rightarrow BLUE \rightarrow GREEN \rightarrow ORANGE \rightarrow RED$, where *BLACK* corresponds to the slowest speed, and *RED* corresponds to the fastest.

We will start with creating the class Particle. Since a particle has a circular shape, it is clear that the particle can be represented by the *x*– and *y*–coordinates of its center, a radius, and *x*– and *y*–coordinates of its velocity.
Therefore the class Particle will have the following instant fields:

```
    double px, py ;          // coordinates of the position
    double vx, vy ;          // coordinates of the velocity
    double r ;               // size of the particle
```

Since the color of the particle corresponds to its speed, we do not need the instant field that would hold the current color of the particle. Instead we will use an array to represent the color scheme that will allow us to assign a color to a particle dynamically:

```
static public final int MAX_VELOCITY = 800 ;        // the upper limit of the velocity in each direction,
                                                    // in pixels per second,
static private Color [] colors = new Color[]
        {  Color.BLACK, Color.BLUE, Color.GREEN, Color.YELLOW, Color.RED  };
```

The source code of the class Particle is placed below.
The purposes of the method *changeSpeed(double fx, double fy),* the method *display(Graphics g)* and *the method move(double time, int left, int right, int top, int bottom)* are self explanatory.
The parameters *left, right, top,* and *bottom* are the boundaries of the Panel.

```
// File Particle .java
import java.awt.* ;
public class Particle
{
        // common fields of the class particle
        static private Color [] colors =  {  Color.BLACK,  Color.GRAY,      Color.BLUE,
                                             Color.GREEN,  Color.YELLOW,  Color.RED };
        static public final double MAX_VELOCITY  =  800.0 ;

        // instant fields
        double px, py ;          // coordinates of the position
        double vx, vy ;          // coordinates of the velocity
        double r ;               // size of the particle

        // constructor
        public Particle( double posx, double posy, double v, double rad)
        {       px = posx ;
                py = posy ;
                double angle     = 2 * Math.PI * Math.random() ;        // chose a direction at random
                vx = v * Math.cos(angle) ;
                vy = v * Math.sin(angle) ;
                r   = rad ;
        }
```

```
// public methods
public void changeSpeed( double fx, double fy )                    // fx and fy are coefficients
{       vx = vx * fx ;
        vy = vy * fy ;

        // the next two lines do not allow the components of the speed exceed MAX_VELOCITY
        if ( Math.abs(vx) > MAX_VELOCITY )   vx = MAX_VELOCITY * Math.signum( vx ) ;
        if ( Math.abs(vy) > MAX_VELOCITY )   vy = MAX_VELOCITY * Math.signum( vy );
}

public void move( double time, int left, int right, int top, int bottom )
{       // find the next position of the particle
        px = px  + vx * time ;
        py = py  + vy * time ;

        // if the particle's position is beyond the boundaries of the panel, the particle has to bounce
        while ( px < left + r || px > right – r )          // while the particle is beyond the vertical walls
        {       vx = -vx  ;                                // change the direction of the horizontal component
                                                           //  of the velocity
                if ( px < left + r )                       // if the particle is beyond the left wall
                        px = 2*(left + r) - px ;           // bounce the particle off the left wall
                else
                        px = 2*(right - r) - px ;          // otherwise, bounce the particle off the right wall
        }

        while ( py < top + r || py > bottom - r )          // while the particle is beyond the horizontal walls
        {       vy = – vy ;                                // change the direction of vertical component
                                                           // of the velocity
                if ( py > bottom - r )                     // if the particle is lower than the bottom
                        py = 2*(bottom - r) - py ;         // then bounce the particle off the bottom wall
                else
                        py = 2*(top + r) - py ;            // otherwise, bounce the particle off the top wall
        }
        return ;
}
```

```
    public void display( Graphics g )
    {
        Color tmp = g.getColor() ;        // store the current color of the Graphics
        int index = (int)( colors.length  * Math.sqrt( vx*vx+vy*vy ) / MAX_VELOCITY ) ;

        if ( index >= colors.length )
            index = colors.length – 1 ;

        g.setColor( colors[ index ] ) ;
        g.fillOval( (int)(px - r) , (int)(py - r), (int)(2*r), (int)(2*r)  ) ;

        g.setColor( tmp ) ;                // reset the current color of the Graphics
    }
}
```

Notice that the class Particle does not "know" anything about how the motion of the particle is scheduled.
Now we will create the panel that simulates the motion. Notice that the variables *tm* and *motion* are declared in the constructor *SimpleMotion*(), since we are not going to reschedule the tasks in this program.

```
// File SimpleMotion.java
import java.awt.* ;
import java.awt.event.* ;
import java.util.*  ;

public class SimpleMotion extends Panel implements KeyListener
{
    // instant fields
    private Particle    prt     = new Particle( 100, 100, 50, 40 ) ;    // particle
    private Dimension   d       = null   ;      // dimension of the panel
    private long        delay   = 1000 ;        // delay before the particle starts moving, in milliseconds
    private long        period  = 100    ;      // frequency of moves, in milliseconds
    private double      coef    = 1.2    ;      // factor of increase/decrease of speed

    // constructor
    public SimpleMotion()
    {
        addKeyListener( this ) ;
        Timer tm = new Timer() ;
```

```
        TimerTask motion = new TimerTask()
        {       public void run()
                {       if ( d != null )
                        {       prt.move( period / 1000.0 ,   0,  d.width, 0, d.height ) ;
                                repaint() ;
                        }
                }
        } ;
        tm.scheduleAtFixedRate( motion, delay, period ) ;
}

// public methods
public void paint( Graphics g )
{
        d = getSize() ;             // obtain the current dimensions of the Panel
                                    // notice that d is used in the method run()
        prt.display( g) ;
}

public void keyPressed( KeyEvent ke)
{
        switch ( ke.getKeyCode() )
        {
                case KeyEvent.VK_UP          :      prt.changeSpeed( 1 , coef ) ;      // increase vy only
                                                    break ;
                case KeyEvent.VK_DOWN        :      prt.changeSpeed( 1, 1/ coef ) ;    // decrease vy only
                                                    break ;
                case KeyEvent.VK_LEFT        :      prt.changeSpeed( 1/ coef, 1) ;     // decrease vx only
                                                    break ;
                case KeyEvent.VK_RIGHT       :      prt.changeSpeed(coef, 1 ) ;         // increase vx only
                                                    break ;
                case KeyEvent.VK_PAGE_UP:           prt.changeSpeed(coef, coef) ;       // increase both
                                                    break ;                             // vx and vy
                case KeyEvent.VK_PAGE_DOWN:         prt.changeSpeed(1/coef, 1/coef) ; // decrease both
                                                    break ;                             // vx and vy
        }
}
```

```
    public void keyReleased( KeyEvent ke) {}

    public void keyTyped ( KeyEvent ke) {}

} // the end of class SimpleMotion
```

Exercises:

1. Compile and run the class SimpleMotion shown above, and then change the code of the class SimpleMotion to implement buffered painting and program to eliminate possible blinking.

2. Write a class *MovingEyes* shows an object of the class *PairOfEyes* moving within the program window and bouncing off the walls. While moving, a *PairOfEyes* should constantly look at the mouse. You can set a speed of the object to a constant absolute value, yet a random direction. The class should employ buffered painting to eliminate possible blinking.

3. Write a class *MovingClock* that would display a round clock that initially appears in the middle of the program window, and a second later the clock starts moving in a random direction at a certain fixed speed and bouncing off the walls. The clock would be an object of an auxiliary class *Clock* that implements all features of the clock. The clock should have an hour arm, a minute arm, and a second arm. The clock should constantly show a correct local time.

4. Suppose you need to write a class that would allow a user to create either a few Clocks or a few PairsOfEyes by clicking a mouse at different points. Newly created objects would start moving at a fixed speed, but in a random direction, and would bounce off the walls. What classes would you design to implement this task?
   Hint: Think about making changes in the programs in the future.