

DESCRIPTION OF THE GAME ENVIRONMENT

A.1 OVERVIEW OF THE GAME ENVIRONMENT

The game environment is comprised of the interface `Drawable` and the two classes `DrawableAdapter` and `GameBoard`. To use the game environment in an application, the interface and these two classes must be included as part of the application (see Appendix B).

Figure A.1 shows a Java application that displays the game environment's window shown in Figure A.2. It assumes the game environment package `edu.sjcny.gpv1` has been added to the system's CLASSPATH variable. If the alternate approach described in Appendix B and in the *IDE Tools* subfolder contained on the book's DVD, which does not require a change in the system's CLASSPATH variable, was used to incorporate the game environment into the application's project, the import statement may not be necessary.

```
1  import edu.sjcny.gpv1.*; //May not be necessary
2  public class GameWindowDemo extends DrawableAdapter
3  {
4      static GameWindowDemo ga = new GameWindowDemo();
5      static GameBoard gb = new GameBoard(ga, "The Game's Title");
6
7      public static void main(String[] args)
8      {
9          showGameBoard(gb);
10     }
11 }
```

Figure A.1

The application **GameWindowDemo** that displays the game environment window.

As shown on line 2 of Figure A.1, game programs that use the game environment must extend the class `DrawableAdapter` and declare a static class level instance of the application's class using the default (no-parameter) constructor, as shown on line 4. Then, an instance of the class `GameBoard` is declared, passing the constructor the class level instance of the application's class and the title of the game (line 5). Finally, the `showGameBoard` method in the `DrawableAdapter` class is invoked from within the `main` method (line 9) and passed the `GameBoard` object declared on line 5. This method displays the application's window shown in Figure A.2.

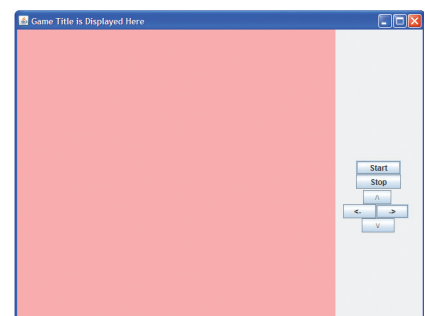


Figure A.2

A game application's window.

The Game Window

As shown in Figure A.2, a game application's window contains six buttons on its right side. The large pink panel to the left of the buttons is called the game board. Game piece objects are displayed on the game board. The color of the game board can be changed by invoking the API `Component` class's `setBackground` method on the `GameBoard` object (declared on line 5 of Figure A.1) and passing it the new board color (an instance of an API `Color` class object).

Timers and Timer Methods

In addition to the six buttons, a `GameBoard` object has three timers associated with it. These begin ticking when the game's player clicks the game window's Start button, and they stop ticking when the game window's Stop button is clicked. The `GameBoard` class contains methods (subprograms) that the programmer can invoke to stop and start a timer, and a method to change the rate at which a timer ticks. These methods are invoked on the `GameBoard` object declared on line 5 of Figure A.1. Section A.2.1 gives the signatures and a description of each of these methods. By default, the tick rates of the timers (named 1, 2, and 3) are once every second, once every half second, and once every quarter second, respectively.

Each timer has a call back method, or subprogram, associated with it, which the game environment invokes every time the timer ticks. The names of the methods, which are described in Section A.2.3, are `timer1`, `timer2`, and `timer3`. The class `DrawableAdapter` contains empty implementations of the methods. If the programmer includes (overrides) these methods in a game program, the game environment will invoke (or "call back") the programmer's versions of the methods after every tick of the timers. Java code placed inside of these methods can be used to keep track of a game's time and to animate objects on the game board.

Game Player Action Methods

There are eight other call back methods in the game environment, which are described in Section A.2.3. Seven of these are invoked by the game environment when the game player performs input actions common to most games. Four of these are associated with the game window's left, right, up, and down buttons, one is associated with the keyboard, and two are associated with the mouse. Empty implementations of the methods are coded in the class `DrawableAdapter`. If the programmer includes (overrides) these methods in a game program, the game environment invokes (calls back) the programmer's overridden version of the methods every time the game player clicks a directional button, presses a keyboard key, or drags or clicks the mouse on the game board.

The eighth call back method in this group of methods is associated with redrawing the game window. It is called by the game environment every time the game window needs to be redrawn (e.g., is minimized and then restored) and every time the seven game player input action call back methods or the three timer call back methods complete their execution. An empty implementation of this method is also included in `DrawableAdapter` class.

Game Window Coordinate System

The game window has a Cartesian coordinate system associated with it as shown in Figure A.3. The coordinate system's origin is in the upper left corner of the game window, with the positive x direction to the right and the positive y direction downward. The height and width of the top and left borders of the window place the upper left corner of the game board at (5, 30). The game window can be sized by using the last two parameters of the game board's four-parameter constructor to specify the coordinates of the lower right corner of the game board, which defaults to (500, 500).

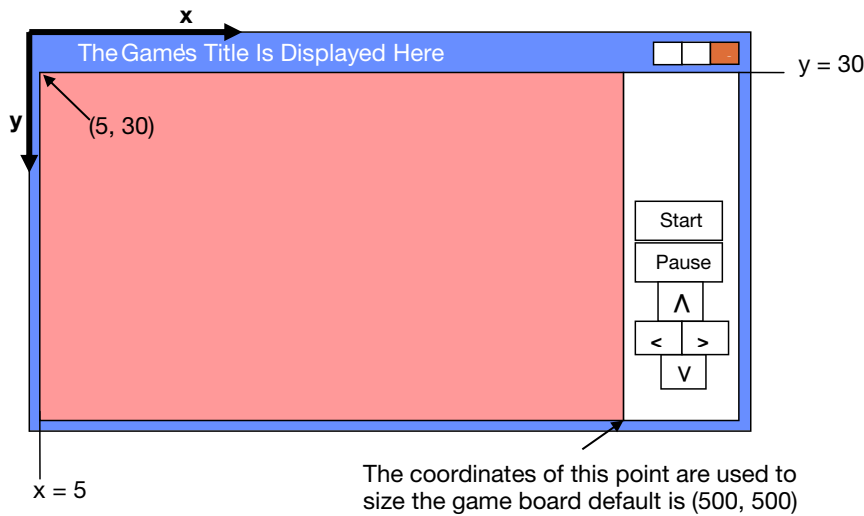


Figure A.3

The game environment coordinate system.

A.2 DESCRIPTION OF THE GAME ENVIRONMENT'S CLASS AND INTERFACE

The game environment is comprised of two classes, named `GameBoard` and `DrawableAdapter`, and one interface named `Drawable`.

A.2.1 The GameBoard Class

The class `GameBoard` contains two constructors. One creates a default-sized game board with a programmer specified title, and the other adds the ability to create a game board and specify its size. A game application must construct a static `GameBoard` object and pass it a static instance of the application's class (see lines 4 and 5 of Figure A.1). Assuming the static instance of the application class was named `ga`, the following line of code creates the `GameBoard` instance `gb` with a default game board size, 500 x 500 pixels:

```
static GameBoard gb = new GameBoard(ga, "The Game's Title");
```

The `GameBoard` object can be used within the game application class to invoke the class's other three methods that are used to set the time increment of any of the `GameBoard` object's three timers and to start and stop these timers. The following invocation stops timer 2:

```
gb.stopTimer(2);
```

Constructors

```
public GameBoard(Object app, String windowTitle)
```

This method constructs a `GameBoard` object, defaulting to the coordinates of the lower right corner of the game board to (500, 500).

Parameters:

`app` is an instance of the application's class.

`windowTitle` will be displayed as the title of the application's window in its title bar.

```
public GameBoard(Object app, String windowTitle,  
                  int xMaxValue, int yMaxValue)
```

This method constructs a `GameBoard` object whose lower-right corner is specified by the last two arguments passed to it.

Parameters:

`app` is an instance of the application's class.

`windowTitle` will be displayed as the title of the application's window in its title bar.

`xMaxValue` is the x-pixel coordinate of the lower-right corner of the game board.

`yMaxValue` is the y-pixel coordinate of the lower-right corner of the game board.

Methods

```
public void setTimerInterval(int timerNumber, int interval)
```

This method sets the interval of one of the game environment's three timers. The default increments for the timers are 1000ms (1 second) for timer 1, 500ms (1/2 second) for timer 2, and 250 ms (1/4 second) for timer 3.

Parameters:

`timerNumber` is the number of the timer (1, 2, or 3) whose interval is being set.

`interval` is the time between ticks of the timer in milliseconds (e.g., 1000 = 1 second).

```
public void startTimer(int timerNumber)
```

This method starts the timer whose number (1, 2, or 3) is passed to it. While the timer is ticking, the timer's call back method (`timer1`, `timer2`, or `timer3`) will be executed on each subsequent tick of the timer. Ticking is stopped when the game player clicks the game board's Pause button or when the `GameBoard` class's `stopTimer` method is invoked. Ticking is restarted when the Start button is clicked or this method is reinvoked.

Parameters:

`timerNumber` designates the timer to be started: 1, 2, or 3.

```
public void stopTimer(int timerNumber)
```

This method stops the timer whose number (1, 2, or 3) is passed to it. After this method is invoked, clicking the game board's Start button will *not* restart the timer. The timer's call back method (`timer1`, `timer2`, or `timer3`) will not be executed until the timer is started again via an invocation of the `startTimer` method, which will also reactivate the game board's Start button.

Parameters:

`timerNumber` designates the timer to be stopped: 1, 2, or 3.

A.2.2 The DrawableAdaper Class

A game program's class must extend the class `DrawableAdaper` (as on line 2 of Figure A.1). It provides empty implementations of the eleven call back methods defined in the game package interface `Drawable`. A description of these eleven methods is given in Section A.2.3. In addition, it provides a method that displays an instance of a `GameBoard` object passed to it (line 9 of Figure A.1).

Methods of the Class DrawableAdapter

```
public static void showGameBoard(GameBoard gb)
```

This method displays the game program's window. Normally, it is invoked as the last statement in the game program's `main` method.

Parameters:

`gb` is the application's `GameBoard` object.

A.2.3 The Interface Drawable

The interface `Drawable` defines eleven call back methods invoked by the game environment. They are coded (as required) in the game application's class (the class that contains the method `main`). They are used to service various actions by the game's player (e.g., a mouse click or drag, a keystroke, or a button click), and to perform processing such as animation every time a game environment's timer ticks.

The call back method `draw` is invoked by the game environment when the game window has to be redrawn (e.g., it is dragged to a new location) and after any of the other ten call back methods complete their execution.

Call Back Methods

public void draw(Graphics g)

This method is invoked when the game application window is initially displayed or needs to be redisplayed and each time one of the other ten call back methods complete their execution.

Parameters: g is an instance of the API class Graphics attached to the game board, which is passed into this method when it is invoked by the game environment. It can be used to draw two-dimensional shapes on the game board by invoking the methods in the Graphics class.

public void timer1()

This method is invoked every time timer 1 ticks. The timer ticking can be started or stopped by invoking the GameBoard class's startTimer and stopTimer methods, respectively. If the timer is ticking, it is stopped whenever the Stop button in the game's window is clicked and restarted whenever the Start button in the game's window is clicked.

public void timer2()

This method is invoked every time timer 2 ticks. The timer ticking can be started or stopped by invoking the GameBoard class's startTimer and stopTimer methods, respectively. If the timer is ticking, it is stopped whenever the Stop button in the game's window is clicked and restarted whenever the Start button in the game's window is clicked.

public void timer3()

This method is invoked every time timer 3 ticks. The timer ticking can be started or stopped by invoking the GameBoard class's startTimer and stopTimer methods, respectively. If the timer is ticking, it is stopped whenever the Stop button in the game's window is clicked and restarted whenever the Start button in the game's window is clicked.

public void leftButton()

This method is invoked whenever the Left button in the game's window is clicked.

public void rightButton()

This method is invoked whenever the Right button in the game's window is clicked.

public void upButton()

This method is invoked whenever the Up button in the game's window is clicked.

public void downButton()

This method is invoked whenever the Down button in the game's window is clicked.

public void keyStruck(char key)

This method is invoked whenever a key on the keyboard is struck. If the key is held down, the method is continually invoked until the key is released.

Parameters: `key` contains the upper case version of the character that was struck. The cursor control keys return 'L', 'R', 'U' or 'D' when the left, right, up, or down arrows are struck.

```
public void mouseClicked(int x, int y, int buttonPressed)
```

This method is invoked whenever a mouse button is clicked.

Parameters: `x` and `y` are the game board coordinates of the mouse cursor location at the time the mouse was clicked.

`buttonPressed` contains a 1 if the left mouse button was clicked or a 3 if the right mouse button was clicked.

```
public void mouseDragged(int x, int y)
```

This method is continually invoked while the mouse is being dragged.

Parameters: `x` and `y` are the game board coordinates of the mouse cursor location at the time the method is invoked.

