```java
package game;

import game.graphic.GraphicsHelper;
import game.gui.*;
import game.highscore.HighScoresManager;
import game.network.client.*;
import game.network.packet.InvitationPacket;
import game.util.Logger;
import game.util.ResourceManager;

import java.awt.*;
import java.awt.event.*;

import javax.swing.*;

/**
 * The <code>GameMenu</code> hold the main method of the game.
 * The application starts from the menu represented by this
 * class. From here the player can start a single or network
 * game, log on to the server, signup and view the high scores;
 */
public class GameMenu extends JFrame implements ActionListener {

    private NetworkManager networkManager;
    private HighScoresManager highScoresManager;

    // GUI components
    private JPanel guiPanel;
    private JButton startButton, multiplayerButton,
      loginoutButton, exitButton, signupButton, highScoresButton;

    // Game menu dialogs
    private LoginDialog loginDialog;
    private SignupDialog signupDialog;
    private OKDialog okDialog;
    private AvailablePlayersDialog availablePlayersDialog;
    private HighScoresDialog highScoresDialog;
    private InvitationDialog invitationDialog;

    private Long sessionId = null;  // The network session id

    private boolean exited = false;
    private boolean startSingle = false;
    private boolean startNetworkGame = false;

    /**
     * Start the game.
     * @param args  No args
     */
    public static void main(String[] args) {
        Logger.init(args);
        GameMenu game = new GameMenu();
        game.start();
    }

    /**
     * Initialize the various objects and set up the GUI.
     */
    public GameMenu() {
        super("Super Game Menu");

        this.addWindowListener(
            new WindowAdapter() {
                public void windowClosing(WindowEvent e) {
                    exitGame();
                }
            }

        );

        // Initialize the managers
        networkManager = new J2EENetworkManager(this);
        highScoresManager = new HighScoresManager(10);
        highScoresManager.setNetworkManager(networkManager);

        // Create the menu GUI
        createGUI();

        // Create game dialogs
        loginDialog = new LoginDialog(this);
        signupDialog = new SignupDialog(this);
        okDialog = new OKDialog(this);
        availablePlayersDialog = new AvailablePlayersDialog(this);
        highScoresDialog = new HighScoresDialog(this, true, highScoresManager);
        invitationDialog = new InvitationDialog(this, false, networkManager);

        this.setSize(230, 300);

        // Center the frame on the screen
        Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
        Dimension windowSize = this.getSize();
        this.setLocation(
                Math.max(0,(screenSize.width - windowSize.width) / 2),
                Math.max(0,(screenSize.height - windowSize.height) / 2));

        this.setVisible(true);

    }

    /**
     * Run in the loop until state changed to start
     * a new game or exit the game
     */
    public void start() {

        while(!exited) {
            try {
                if (startSingle) {
                    startSinglePlayerGame();
                } else if (startNetworkGame){
                    startNetworkGame();
                } else {
                    Thread.sleep(50);
                }
            }
            catch (InterruptedException ie) {
                // ignore
            }
        }
        exitGame();

    }

    /**
     * Start a single player game.
     */
    public void startSinglePlayerGame() {
        startSingle = false;
        startGame(false, true);
    }

    /**
     * Start a network game.
     */
    public void startNetworkGame() {
        startNetworkGame = false;
        // Start new network game. The inviter is the controller
        startGame(true, networkManager.isInviter());

    }

    /**
     * Start a new game.
     * @param networkGame True if starting a network game
     * @param controller  True is this machine is the controller
     */
    public void startGame(boolean networkGame, boolean controller) {
        // Hide the game menu
        this.setVisible(false);

        if (sessionId != null) {
            // Don't accept invitations while playing
            try {
                networkManager.acceptInvitations(false);
            }
            catch (NetworkException ne) {
                Logger.exception(ne);
            }
        }

        GameNetworkManager gnm = null;
        if (networkGame){
            try {
                gnm = networkManager.getGameNetworkManager();
            }
            catch (NetworkException ne) {
                Logger.exception(ne);
                return; // don't start the game
            }
        }

        GameLoop gameLoop = new GameLoop(networkGame, controller,
                highScoresManager, gnm);

        Thread gameThread = new Thread(gameLoop);
        gameThread.run();
        try {
            // Join the game loop and wait until it's finished
            gameThread.join();
        }
        catch (InterruptedException ie) {
            Logger.exception(ie);
        }

        // Game finished
        if (sessionId != null) {
            // Accept invitations
            try {
                networkManager.acceptInvitations(true);
            }
            catch (NetworkException ne) {
                Logger.exception(ne);
            }
        }

        this.requestFocus();

        // Show the game menu
        this.setVisible(true);
    }

    /**
     * Setup the GUI for the game menu.
     */
    public void createGUI() {

        if (!Logger.isDebug()) {
            this.setUndecorated(true);
        }

        Container container = this.getContentPane();

        guiPanel = new JPanel(new GridLayout(8, 1, 5, 0));

        JLabel menuLabel = new JLabel("Game Menu", SwingConstants.CENTER);
        menuLabel.setFont(ResourceManager.getFont(Font.BOLD, 16));
        menuLabel.setForeground(Color.BLUE);
        guiPanel.add(menuLabel);

        startButton = createButton("Start Single Player",
                "Starts a new single player game");

        multiplayerButton = createButton("Start Multiplayer Game",
                "Starts a new multiplayer game");
        multiplayerButton.setEnabled(false);

        loginoutButton = createButton("Login", "Login");

        signupButton = createButton("Signup", "Signup");

        highScoresButton = createButton("High Scores", "View High Scores");

        exitButton = createButton("Exit", "Exit the game");

        guiPanel.add(startButton);
        guiPanel.add(multiplayerButton);
        guiPanel.add(loginoutButton);
        guiPanel.add(signupButton);
        guiPanel.add(highScoresButton);
        guiPanel.add(exitButton);
```

```java
        guiPanel.setBackground(Color.BLACK);

        container.add(guiPanel, BorderLayout.CENTER);

        this.validate();

    }

    /**
     * Creates and returns a customized button.
     * @param buttonText  Text to display on the button
     * @param toolTipText Button tooltip text
     * @return  Customized button.
     */
    private JButton createButton(String buttonText,
            String toolTipText) {
        JButton button = new JButton();
        button.setToolTipText(toolTipText);
        button.addActionListener(this);

        // Change the button look
        customizeButton(button, buttonText);

        return button;
    }

    private void customizeButton(JButton button,
            String buttonText) {
        // Load the base image for the button
    Image srcImage = ResourceManager.loadImage(
            GameConstants.IMAGES_DIR + GameDialog.BTN_BIG_IMAGE);

    int width = srcImage.getWidth(null);
    int height = srcImage.getHeight(null);

    Font menuFont = ResourceManager.getFont(16);

    // Get a compatible translucent image
    Image image = GraphicsHelper.getCompatibleImage(this,
            width, height, Transparency.TRANSLUCENT);
        // Draw the source image and the button text
    // on the tranlucent image with alpha composite of 0.8
    Graphics2D g = (Graphics2D)image.getGraphics();
        Composite alpha = AlphaComposite.getInstance(
            AlphaComposite.SRC_OVER, 0.8f);
        g.setComposite(alpha);
        g.drawImage(srcImage, 0, 0, null);
        g.setFont(menuFont);
        GraphicsHelper.drawInMiddle(g, image, buttonText);
        g.dispose();

        // Create an image icon for the default button image
        ImageIcon iconDefault = new ImageIcon(image);

        // Create a pressed image
        image = GraphicsHelper.getCompatibleImage(this,
            width, height, Transparency.TRANSLUCENT);
        g = (Graphics2D)image.getGraphics();
        alpha = AlphaComposite.getInstance(
                AlphaComposite.SRC_OVER, 0.9f);
        g.setComposite(alpha);
        // a bit lowered and to the right button
        g.drawImage(srcImage, 2, 2, null);
        g.setFont(menuFont);
        GraphicsHelper.drawInMiddle(g, image, buttonText);
        g.dispose();
        ImageIcon iconPressed = new ImageIcon(image);

        // Create disabled button image
        image = GraphicsHelper.getCompatibleImage(this,
            width, height, Transparency.TRANSLUCENT);
        g = (Graphics2D)image.getGraphics();
        alpha = AlphaComposite.getInstance(
                AlphaComposite.SRC_OVER, 0.3f);
        g.setComposite(alpha);
        g.drawImage(srcImage, 0, 0, null);
        g.setFont(menuFont);
        GraphicsHelper.drawInMiddle(g, image, buttonText);
        g.dispose();
        ImageIcon iconDisabled = new ImageIcon(image);

        button.setOpaque(false);
        button.setFocusPainted(false);
        button.setFocusable(false);
        button.setContentAreaFilled(false);
        button.setBorderPainted(false);
        button.setMargin(new Insets(0, 0, 0, 0));
        button.setIcon(iconDefault);
        button.setPressedIcon(iconPressed);
        button.setDisabledIcon(iconDisabled);
        button.setSize(button.getPreferredSize());

    }

    /**
     * Show the login dialog.
     */
    private void popLoginDialog() {
        loginDialog.popDialog();
    }

    /**
     * Show the signup dialog.
     */
    private void popSignupDialod() {
        signupDialog.popDialog();
    }

    /**
     * Show the high scores dialog.
     */
    private void popHighScoresDialog() {
        highScoresDialog.popDialog();
    }

    /**
     * Show the available players dialog.
     */
    private void popAvailablePlayersDialog() {
        availablePlayersDialog.popDialog();;
    }
    public void actionPerformed(ActionEvent event) {
      Object eventSource = event.getSource();
        if (eventSource == exitButton) {
            exited = true;

        } else if (eventSource == loginoutButton) {
            if (sessionId == null) {
                // User not logged in yet
                popLoginDialog();
            }
            else {
                // User logged in so we log out
                logout();
            }

        } else if (eventSource == startButton) {
            startSingle = true;

        } else if (eventSource == signupButton) {
            popSignupDialod();

        } else if (eventSource == multiplayerButton) {
            popAvailablePlayersDialog();

      } else if (eventSource == highScoresButton) {
          popHighScoresDialog();
      }
  }

    /**
     * Called by the login and sigup dialogs when the user succesfully
     * logs in.
     * @param sessionId Session id of the user
     */
    public void setLoggedUser (Long sessionId) {
        try {
          this.sessionId = sessionId;
          customizeButton(loginoutButton, "Logout");
          loginoutButton.setToolTipText("Logout");
          // Enable the multiplayer button
          multiplayerButton.setEnabled(true);
          // Accept game invitations
          networkManager.acceptInvitations(true);
        }
        catch (NetworkException ne) {
            Logger.showErrorDialog(this, ne.getMessage());
        }
    }

    /**
     * Called if the user clicks on the logout button and by
     * the signup dialog if the signup completed successfully
     * before the user logs in with the new user name.
     */
    public void logout() {
        if (sessionId != null) {
            try {
              networkManager.acceptInvitations(false);
              networkManager.logout();
              System.out.println("Logged out successfully");
            }
            catch (NetworkException ne) {
                System.err.println(ne.getMessage());
            }
            finally {
                customizeButton(loginoutButton, "Login");
              loginoutButton.setToolTipText("Login");
              multiplayerButton.setEnabled(false);
                this.sessionId = null;
            }
        }
    }

    /**
     * Called when the user exites the game or closes
     * the window.
     */
    private void exitGame() {
        finalizeGame();
        System.exit(0);
    }

    /**
     * Releases any resources and updates the server if
     * logged in.
     */
    private void finalizeGame() {
        if (sessionId != null) {
            try {
                networkManager.acceptInvitations(false);
                networkManager.logout();
            }
            catch (NetworkException ne) {
                System.err.println(ne.getMessage());
            }
        }
    }

    /**
     * Returns the network manager.
     * @return The network manager.
     */
    public NetworkManager getNetworkManager() {
        return this.networkManager;
    }

    /**
     * This method is called when network user accepts the invitation
     * to play or the local user accepts the invitation to play.
     * @param start True to start a network game.
     */
    public void setStartMultiplayer(boolean start) {
        this.startNetworkGame = start;
    }

    /**
     * Invoked by the network manager when a reply to a previous invitation
     * to play sent by this user arrived.
     * @param accepted  True is the player accepted the invitation.
     * @param userName  Name of the player
     */
    public void invitationAccepted(boolean accepted, String userName) {
        if (accepted) {

            okDialog.setText("User " + userName +
                    " accepted your invitation");
            okDialog.popDialog();

            availablePlayersDialog.hideDialog();

            // Start a new multiplayer game
            setStartMultiplayer(true);

        }
        else {

            okDialog.setText("User " + userName +
                    " rejected your invitation");
            okDialog.popDialog();

            availablePlayersDialog.setStatusText("User " +
                        userName + " rejected your invitation");
        }

        // reset the invitation status in the dialog
        availablePlayersDialog.reset();

    }

    /**
     * Invoked by the network manager when an invitation to play arrives
     * from an online player.
     * @param invitation  Invitation packet with the invitation details
     */
    public void invitationArrived(InvitationPacket invitation) {

        invitationDialog.invitationArrived(invitation);

    }

    /**
     * Invoked by the network manager when a previous invitation
     * to play was cancelled by the inviter.
     */
    public void invitationCancelled() {
        invitationDialog.invitationCancelled();
    }

}
```

```java
package game;

import java.awt.*;
import java.lang.reflect.InvocationTargetException;

import javax.swing.*;

/**
 * The <code>GUIManager</code> is a helper class to make the Swing
 * components work with the active rendering used in the game when it
 * is running (not in the game menu).
 * This class installs a <code>RepaintManager</code> that ignores
 * the repaints from swing components so they won't interrupt the
 * game. We call the <code>render</code> method of this class when we
 * want to render game dialogs (which extend <code>JPanel</code>).
 */
public class GUIManager {

    private GameLoop gameLoop;
    private ScreenManager screenManager;
    private RepaintManager oldRepaintManager;

    public GUIManager(GameLoop gameLoop, ScreenManager screenManager) {

        this.gameLoop = gameLoop;
        this.screenManager = screenManager;

        // Save the current RepaintManager to restore later
        oldRepaintManager = RepaintManager.currentManager(null);
        // Set new RepaintManager that ignores repainting
        RepaintManager.setCurrentManager(new NullRepaintManager());

        JFrame gameFrame = screenManager.getFullScreenWindow();

        Container container = gameFrame.getContentPane();
          ((JComponent)container).setOpaque(false);

        gameFrame.validate();
    }

    /**
     * Render the layered pane and all its sub components.
     */
    public void render(final Graphics g) {

        final JFrame gameFrame = (JFrame)screenManager.getFullScreenWindow();

        // Use the EventQueue.invokeAndWait to prevent deadlocks
        if (!SwingUtilities.isEventDispatchThread()) {
          try {
              EventQueue.invokeAndWait(
                  new Runnable() {
                      public void run() {
                          gameFrame.getLayeredPane().paintComponents(g);
                      }
                  }
              );
          }
          catch (InterruptedException ex) {
              // Ignore
```

```
            }
            catch (InvocationTargetException  ex) {
                // Ignore
            }
        }
        else {
            gameFrame.getLayeredPane().paintComponents(g);
        }

    }

    /**
     * Adds a dialog to the modal layer pane of the game frame
     * @param dialog Dialog to add
     */
    public void addDialog(JPanel dialog) {
        gameLoop.getScreenManager().getFullScreenWindow().
      getLayeredPane().add(dialog, JLayeredPane.MODAL_LAYER);
    }

    /**
     * Restores the original <code>RepaintManager</code>.
     */
    public void restoreRepaintManager() {
        RepaintManager.setCurrentManager(oldRepaintManager);
    }

    /**
     * We use the NullRepaintManager to disable all the repainting
     * since all the painting is done from the game loop
     */
  private class NullRepaintManager extends RepaintManager {

      public NullRepaintManager() {
          setDoubleBufferingEnabled(false);
      }

      public void addInvalidComponent(JComponent c) {
          // do nothing
      }

      public void addDirtyRegion(JComponent c, int x, int y,
          int w, int h)
      {
          // do nothing
      }

      public void markCompletelyDirty(JComponent c) {
          // do nothing
      }

      public void paintDirtyRegions() {
          // do nothing
      }

  }

}
```

```java
package game;

import java.awt.Dimension;
import java.io.File;
import java.util.*;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;

import org.w3c.dom.*;

import game.ship.*;
import game.util.Logger;

/**
 * The <code>LevelsManager</code> class is used to load the levels
 * from an XML file.
 */
public class LevelsManager {

    private GameLoop gameLoop;

    private Document xmlDocument;
    private int currentLevel = 0;
    private int lastLevel;
    private boolean levelFinished = true;

    /**
     * Construct a LevelManager and load the levels xml file.
     * @param gameLoop  Reference to the game loop
     */
    public LevelsManager(GameLoop gameLoop) {
        this.gameLoop = gameLoop;
        loadXMLFile();
    }

    /**
     * Returns the current level in the game.
     * @return The current level in the game.
     */
    public int getCurrentLevel() {
        return this.currentLevel;
    }

    /**
     * Returns true if the current level is the last one.
     * @return  True if the current level is the last level.
     */
    public boolean isLastLevel() {
        return lastLevel == currentLevel;
    }

    /**
     * This method is called to inform the level manager
     * to update to current level. It is called whenever
     * a packet with new level information arrives in a
     * network game.
     */
    public void nextLevel() {
        currentLevel++;
    }

    /**
     * Loads the next level from the xml file and returns the enemy
     * ships map.
     * @return  Map with the enemy ships for the level.
     */
    public Map loadNextLevel() {
        currentLevel++;
        return loadLevel(currentLevel);
    }

    /**
     * Loads the level number <code>levelNumber</code> from the
     * levels xml file.
     * @param levelNumber Number of the level to load.
     * @return  Map of enemy ships
     */
    public Map loadLevel(int levelNumber) {

        int curObjectID = GameConstants.FIRST_ENEMY_SHIP_ID;
        Map enemyShips = new HashMap();

        Element level = getLevelElement(levelNumber);

        if (level == null) {
            throw new RuntimeException("Error loading the level" + levelNumber
                    + "from file");
        }

        // Load the background image of the current level
        loadLevelBGImage(level);

        Dimension screenDimention =
            gameLoop.getScreenManager().getScreenDimension();

        // Get and create the ship types and ammount for the level
        NodeList ships = level.getElementsByTagName("enemyShips");
        for (int i = 0; i < ships.getLength(); i++) {

            Element enemyShipsNode = (Element) ships.item(i);
            Node shipTypeNode =
                enemyShipsNode.getElementsByTagName("shipType").item(0);
            Node numOfShipsNode =
                enemyShipsNode.getElementsByTagName("numberOfShips").item(0);

            String typeStr = shipTypeNode.getFirstChild().getNodeValue();
            String numShipsStr = numOfShipsNode.getFirstChild().getNodeValue();

            int shipType = Integer.parseInt(typeStr);
            int numOfShips = Integer.parseInt(numShipsStr);

            // Create the ship objects
            for (int j = 0; j < numOfShips; j++, curObjectID++) {
                enemyShips.put(new Integer(curObjectID),
                  new EnemyShip(curObjectID, shipType,
                        (float)(50+Math.random()*(screenDimention.width-50)),
                        (float)(50+Math.random()*screenDimention.height/2),
                    ShipProperties.getShipProperties(shipType)));
            }
```

```java
        }

        return enemyShips;

    } // end method loadLevel

    /**
     * Finds and returns the requested level node from the xml file.
     * @param levelNumber Level to load.
     * @return  Element with the level details. Null if not found.
     */
    public Element getLevelElement(int levelNumber) {
        Element level = null;
        boolean levelFound = false;
        // Get all the level nodes
        NodeList levels = xmlDocument.getElementsByTagName("level");

        // Find the level node with id equals to levelNumber
        for(int i = 0; i < levels.getLength() && !levelFound; i++) {

            level = (Element) levels.item(i);
            // Get the attributes list
            NamedNodeMap attributes = level.getAttributes();
            // Get the levelNum attribute
            Node levelNum = attributes.getNamedItem("levelNum");
            if (levelNum.getNodeValue().equals(String.valueOf(levelNumber))) {
                levelFound = true;
            }
        }

        return level;
    }

    /**
     * Search for the backgroung image in the level element.
     * If exists, set the game backgroung image.
     * @param level Elment with the level info
     */
    public void loadLevelBGImage(Element level) {
        Node bgImage = level.getElementsByTagName("backgroundImage").item(0);
        if (bgImage != null) {
            String bgImageName = bgImage.getFirstChild().getNodeValue();
            gameLoop.getStaticObjectsManager().
              setBackgroundImage(bgImageName);
        }
    }

    /**
     * Loads data needed for the local not controller machine in a
     * network game.
     * @param levelNumber Number of the level
     */
    public void loadLocalLevelData(int levelNumber) {
        Element level = getLevelElement(levelNumber);
        if (level != null) {
            loadLevelBGImage(level);
        }
    }


    /**
```

```
     * Loads and parses the levels XML file to the memory. If any error
     * occurs exit with the exeption.
     * Also checks what is the last level in the game.
     */
    private void loadXMLFile() {
        try {
            DocumentBuilderFactory factory =
                DocumentBuilderFactory.newInstance();
            factory.setValidating(true);
            DocumentBuilder builder = factory.newDocumentBuilder();
            this.xmlDocument = builder.parse(
                   new File(GameConstants.CONFIG_DIR+"levels.xml"));

            Element root = xmlDocument.getDocumentElement();

            Node lastLevelNode = root.getElementsByTagName("lastLevel").item(0);
            lastLevel =
Integer.parseInt(lastLevelNode.getAttributes().getNamedItem("levelNum").getNodeVa
lue());
        }
        catch (Exception e) {
            // If any exception occures during the parsing exit the game
            Logger.exception(e);
            System.exit(-1);
        }
    }
}
```

```java
package game;

import java.awt.*;
import java.util.*;

import game.input.InputManager;
import game.network.client.GameNetworkManager;
import game.network.packet.*;
import game.ship.*;
import game.ship.bonus.Bonus;
import game.ship.weapon.*;

/**
 * The <code>PlayerManager</code> manage the local player ship and the
 * network player ship (if one exists).
 * It also implements the <code>ShipContainer</code> interface to allow
 * the ships to communicate with it.
 */
public class PlayerManager implements ShipContainer, PacketHandler {

    private final int handlerID = GameConstants.PLAYER_MANAGER_ID;

    private GameLoop gameLoop;
    private InputManager inputManager;

    private PlayerShip player1Ship, player2Ship, localPlayer, networkPlayer;

    /** Collection of active shots fired by the ships */
    private Collection shots;
    /** Collection of targets for the ships (i.e., enemy ships) */
    private Collection targets;
    /** True if the player ship (and network ship) are destroyed */
    private boolean gameOver;

    /**
     * Construct the PlayerManager. Creates the player ship(s).
     * @param gameLoop  Reference to the game loop
     */
    public PlayerManager(GameLoop gameLoop) {

        this.gameLoop = gameLoop;
        this.inputManager = gameLoop.getInputManager();
        this.shots = new ArrayList();
        this.targets = new ArrayList();
        this.gameOver = false;

        Dimension screen = gameLoop.getScreenManager().getScreenDimension();

        int x = screen.width / 2;
        int y = screen.height - 60;

        player1Ship = new PlayerShip(GameConstants.PLAYER1_ID,
                ShipProperties.PLAYER_SHIP_TYPE_1, x, y,
                ShipProperties.getShipProperties(
                        ShipProperties.PLAYER_SHIP_TYPE_1));

        player1Ship.setShipContainer(this);

        if (gameLoop.isNetworkGame()) {
            x = screen.width / 3;
            player1Ship.setX(x);  // Change player 1 location

            player2Ship = new PlayerShip(GameConstants.PLAYER2_ID,
                    ShipProperties.PLAYER_SHIP_TYPE_2, 2*x, y,
                    ShipProperties.getShipProperties(
                            ShipProperties.PLAYER_SHIP_TYPE_2));

            player2Ship.setShipContainer(this);
        }

        // Set the local player and network player references
        if (gameLoop.isNetworkGame()) {
            if (gameLoop.isController()) {
                localPlayer = player1Ship;
                networkPlayer = player2Ship;
            }
            else {
                localPlayer = player2Ship;
                networkPlayer = player1Ship;
            }
        }
        else {
            localPlayer = player1Ship;
        }

    }

    /**
     * Get ready for a new level. Clear reminders from previous
     * level and set the enemy ships.
     * @param enemyShips  Collection of enemy ships to be used as targets
     */
    public void newLevel(Collection enemyShips) {
        targets.clear();
        shots.clear();
        addTarget(enemyShips);
    }

    /**
     * Adds a new target to the targets collection.
     * @param target  New target to add.
     */
    public void addTarget(Target target) {
        this.targets.add(target);
    }

    /**
     * Adds a collection of targets.
     * @param targets Collection of Target objects.
     */
    public void addTarget(Collection targets) {
        this.targets.addAll(targets);
    }

    /**
     * Returns the local player ship.
     * @return  Local player ship
     */
    public PlayerShip getLocalPlayerShip() {
        return localPlayer;
    }

    /**
     * Returns the network player ship.
     * @return  Network player ship
     */
    public PlayerShip getNetworkPlayerShip() {
        return networkPlayer;
    }

    /**
     * Returns player one ship (the left player)
     * @return  Player 1 ship
     */
    public PlayerShip getPlayer1Ship() {
        return player1Ship;
    }

    /**
     * Returns player 2 ship. Null if there is no player 2 ship.
     * @return  Player 2 ship.
     */
    public PlayerShip getPlayer2Ship() {
        return player2Ship;
    }

    /**
     * Gather the player input (the input collected by the
     * <code>InputManager</code> class).
     */
    public void gatherInput() {

        float oldDx = localPlayer.getDx();
        float oldDy = localPlayer.getDy();

        float velocityX = 0;
        float velocityY = 0;

        if (inputManager.moveLeft.isPressed()) {
            velocityX -= localPlayer.getMaxDX();
        }
        if (inputManager.moveRight.isPressed()) {
            velocityX += localPlayer.getMaxDX();
        }
        if (inputManager.moveUp.isPressed()) {
            velocityY -= localPlayer.getMaxDY();
        }
        if (inputManager.moveDown.isPressed()) {
            velocityY += localPlayer.getMaxDY();
        }

        if (oldDx != velocityX || oldDy != velocityY) {
            localPlayer.setDx(velocityX);
            localPlayer.setDy(velocityY);
            if (gameLoop.isNetworkGame()) {
                // Force the local ship to send packet
                localPlayer.forcePacket();
            }
        }

        if (inputManager.fireBullet.isPressed()) {
            localPlayer.shoot();
        }
    }

    /**
     * Updates the state of all the managed objects.
     * @param elapsedTime Time elapsed since last call to this method
     * in milliseconds.
     */
    public void update(long elapsedTime) {

        player1Ship.update(elapsedTime);
        if (player1Ship.isActive()) {
            fixPlace(player1Ship);
        }

        if (player2Ship != null) {
            player2Ship.update(elapsedTime);
            if (player2Ship.isActive()) {
                fixPlace(player2Ship);
            }
        }

        // Update shots
        Iterator shotsItr = shots.iterator();
        while (shotsItr.hasNext()) {
            Sprite shot = (Sprite) shotsItr.next();
            shot.updatePosition(elapsedTime);
            if (isOutOfScreen(shot)) {
                shotsItr.remove();
            }
        }

        ////////////////////////
        // Process Collisions //
        ////////////////////////


        // Process ship-to-ship collisions
        player1Ship.processCollisions(targets);
        if (player2Ship != null) {
            player2Ship.processCollisions(targets);
        }

        // Process shots to enemy ship collisions
        shotsItr = shots.iterator();
        while (shotsItr.hasNext()) {
            Bullet shot = (Bullet) shotsItr.next();
            shot.processCollisions(targets);
            if (shot.isHit()) {
                shotsItr.remove();
            }
        }

        // Check if the game is over
        if (gameLoop.isNetworkGame()) {
            // Check if game over
            if (getLocalPlayerShip().isDestroyed() &&
                    getNetworkPlayerShip().isDestroyed()) {
                gameOver = true;
            }
        }
        else if (getLocalPlayerShip().isDestroyed()) {
            gameOver = true;
        }

    }

    /**
     * Renders all relevant objects and data (ships, shots, etc.)
     */
    public void render(Graphics g) {
        player1Ship.render(g);

        if (player2Ship != null) {
            player2Ship.render(g);
        }

        // Render shots
        Iterator shotsItr = shots.iterator();
        while (shotsItr.hasNext()) {
            Sprite shot = (Sprite) shotsItr.next();
            shot.render(g);
        }
    }

    /**
     * Fix the ship location and velocity if it is trying to
     * exit the screen bounds.
     * @param ship  Ship to fix its place.
     */
    private void fixPlace(Ship ship) {

        Dimension screenDimention =
            gameLoop.getScreenManager().getScreenDimension();

        Insets insets =
            gameLoop.getScreenManager().getScreenInsets();

        // If the ship exits the screen and still in the wrong
        // direction, change its velocity so it will get back
        if (ship.getX() < insets.left && ship.getDx() < 0) {
            ship.setX(insets.left);
            ship.setDx(0);
        }
        if (ship.getX()+ ship.getWidth() >
                screenDimention.width - insets.right &&
                ship.getDx() > 0) {
            ship.setX(screenDimention.width - ship.getWidth() - insets.right);
            ship.setDx(0);
        }
        if (ship.getY() < insets.top && ship.getDy() < 0) {
            ship.setY(insets.top);
            ship.setDy(0);
        }
        if (ship.getY() + ship.getHeight() >
                screenDimention.height - insets.bottom &&
                ship.getDy() > 0) {
            ship.setY(screenDimention.height -
                    ship.getHeight() - insets.bottom);
            ship.setDy(0);
        }
    }

    /**
     * Return true if the sprite is off the screen bounds.
     * @param sprite  Sptite to test.
     * @return  True if the sprite is off the screen bounds
     */
    private boolean isOutOfScreen(Sprite sprite) {
        Dimension screenDimension =
            gameLoop.getScreenManager().getScreenDimension();

        return sprite.getX() + sprite.getWidth() < 0 ||
          sprite.getX() > screenDimension.width ||
          sprite.getY() + sprite.getHeight() < 0 ||
          sprite.getY() > screenDimension.height;
    }

    /**
     * Inherited from <code>ShipContainer</code> interface.
     * Not in use for the <code>PlayerManager</code>
     */
    public void addShip(Ship ship) {}

    /**
     * Adds shot to the shots collection.
     * @param shot Shot to add.
     */
    public void addShot(Bullet shot) {
        shots.add(shot);
    }

    /**
     * Returns true if this macine is the controller.
     */
    public boolean isController() {
        return gameLoop.isController();
    }

    /**
     * Return true if this is a network game.
     */
    public boolean isNetworkGame() {
        return gameLoop.isNetworkGame();
    }

    /**
     * Returns the game network manager. Null if this is not
     * a network game.
     */
    public GameNetworkManager getNetworkManager() {
        return gameLoop.getGameNetworkManager();
    }

    /**
     * Returns the network handler id of this object.
     */
    public int getHandlerId() {
        return this.handlerID;
    }
```

```java
    /**
     * Handles incoming packets. If the packet should be handle by
     * the maneger than handle it otherwise if the network player
     * ship should to handle it.
     * @param packet Packet to handle
     */
    public void handlePacket(Packet packet) {

        if (packet instanceof BulletPacket) {
            BulletPacket bulletPacket = (BulletPacket)packet;

            BulletModel model = bulletPacket.getBulletModel();

            if (packet.handlerId == getNetworkPlayerShip().getHandlerId()) {

                Bullet bullet = WeaponFactory.getBullet(model,
                        getNetworkPlayerShip());

                addShot(bullet);
            }

            packet.setConsumed(true);

        }
        else if (packet instanceof PlayerQuitPacket) {
            // Network player quit the game. Set this machine as the
            // controller and send no more packets back to the user.
            // Also destroy the network player's ship.
            gameLoop.setController(true);
            gameLoop.setNetworkGame(false);

            getNetworkPlayerShip().destroy();

            packet.setConsumed(true);

        }
        else {
            // Let the ship handle it
            if (packet.handlerId == networkPlayer.getHandlerId()) {
                getNetworkPlayerShip().handlePacket(packet);
            }
        }

    }

    /**
     * Currently this object doesn't creates it's own packets
     */
    public void createPacket(GameNetworkManager netManager) {

    }

    /**
     * Returns the network handler id of this object.
     */
    public boolean isGameOver() {
        return gameOver;
    }

    /**
     * Player ship doesn't release bonuses
     */
    public void addBonus(Bonus bonus) {
        // Player ship doesn't release bonuses
    }

}
```

```java
package game;

import game.graphic.GraphicsHelper;
import game.util.Logger;
import game.util.ResourceManager;

import java.awt.*;
import java.awt.image.BufferStrategy;
import java.awt.image.BufferedImage;
import java.lang.reflect.InvocationTargetException;

import javax.swing.JFrame;

/**
 * The <code>ScreenManager</code> class handles the graphic environment
 * and display settings. It holds the drawing area frame of the game
 * and has some helper methods for grapihcs and images.
 * The screen manager uses a <code>BufferStrategy</code> to manage
 * the double buffering of the game frame (to prevent flickering).
 * The <code>ScreenManager</code> is a singleton so only one object exists.
 */
public class ScreenManager {

    private static ScreenManager screenManager;
    private GraphicsEnvironment ge;
    private GraphicsDevice gd;
    private DisplayMode oldDM;
    private boolean fullScreen;
    private JFrame gameFrame;
    private boolean debugMode;

    /**
     * Private constructor to allow only one instance of the
     * <code>ScreenManager</code> class.
     */
    private ScreenManager() {
      ge = GraphicsEnvironment.getLocalGraphicsEnvironment();
      gd = ge.getDefaultScreenDevice();
      oldDM = gd.getDisplayMode();
      gameFrame = new JFrame();
      debugMode = Logger.isDebug();
    }

    /**
     * Returns the single <code>ScreenManager</code> instance.
     * @return  Screen manager instance
     */
    public static ScreenManager getInstance() {
        if (screenManager == null) {
            screenManager = new ScreenManager();
        }
        return screenManager;
    }

    /**
     * Sets full screen mode with the default display mode (screen
     * resolution of 800x600, bit depth of 32 and the current screen
     * refresh rate.
     *
     */
    public void setFullScreen() {
        DisplayMode displayMode = new DisplayMode(800, 600, 32,
                DisplayMode.REFRESH_RATE_UNKNOWN);
        setFullScreen(displayMode);
    }

    /**
     * Sets full screen mode with the displayMode parameters.
     * If the current machine doesn't support full screen mode
     * we use an undecorated frame with the size 500x600.
     * @param displayMode Full screen display mode.
     */
    public void setFullScreen(DisplayMode displayMode) {

        gameFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        if (!debugMode) {
            gameFrame.setUndecorated(true);
        }
        gameFrame.setIgnoreRepaint(true);
        gameFrame.setResizable(false);
        gameFrame.setFocusable(true);
        gameFrame.requestFocus();

      if (!debugMode && gd.isFullScreenSupported()) {

          gd.setFullScreenWindow(gameFrame);

          if(gd.isDisplayChangeSupported()) {
              gd.setDisplayMode(displayMode);
                // Fix for Mac OS X
                gameFrame.setSize(displayMode.getWidth(),
                        displayMode.getHeight());
          }
      }

      else { // Full screen is not supported

          gameFrame.resize(500,600);
          gameFrame.show();
      }

        // Create a buffer strategy for the game frame
        // Avoid potential deadlock in JDK 1.4
      // The invokeAndWait cannot be called from event dispatcher
      // thread meaning, not as an action of the AWT/Swing
        try {
            EventQueue.invokeAndWait(
                new Runnable() {
                    public void run() {
                        gameFrame.createBufferStrategy(2);
                    }
                }
            );
        }
        catch (InterruptedException ie) {
            Logger.exception(ie);
        }
        catch (InvocationTargetException  ite) {
            Logger.exception(ite);
        }
    }

    /**
     * Returns the game frame window.
     * @return  Game frame
     */
    public JFrame getFullScreenWindow() {
        if (fullScreen) {
            return (JFrame) gd.getFullScreenWindow();
        }
        else {
            return gameFrame;
        }

    }

    /**
     * Exits the full screen mode.
     */
    public void exitFullScreen() {

        if (fullScreen) {
          if (gd.isDisplayChangeSupported()) {
              gd.setDisplayMode(oldDM);
          }

            /** TODO: why is this throwing error? */
            Window window = gd.getFullScreenWindow();
            if (window != null) {
                window.dispose();
            }

          gd.setFullScreenWindow(null);
        }
        else {
            gameFrame.show(false);
            gameFrame.dispose();
        }

    }

    /**
     * Returns the graphics object of the game frame.
     * @return  Graphics object of the game frame.
     */
    public Graphics2D getGraphics() {
        BufferStrategy bs = gameFrame.getBufferStrategy();
        return (Graphics2D) bs.getDrawGraphics();
    }

    /**
     * Show the contents of the game frame. This method should
     * be called after the current frame rendering is finished
     * to display the results.
     */
    public void show() {
        BufferStrategy bs = gameFrame.getBufferStrategy();
        if (!bs.contentsLost()) {
            bs.show();
        }
    }

    /**
     * Returns the game frame screen dimensions.
     * @return The game frame screen dimension.
     */
    public Dimension getScreenDimension() {
        return gameFrame.getSize();
    }

    /**
     * Returns the game frame insets.
     * @return The game frame insets.
     */
    public Insets getScreenInsets() {
        return gameFrame.getInsets();
    }

    /**
     * Creates and returns a compatible image from the image found in
     * <code>imageName</code> image under the images folder. The source
     * image will automatically be resized to the game frame window if it's
     * to big.
     * @param imageName Source image name
     * @param transparency Image transparency
     * @see java.awt.GraphicsConfiguration
     */
    public BufferedImage getCompatibleImage(String imageName, int transparency) {

        Image srcImage = ResourceManager.loadImage(
                GameConstants.IMAGES_DIR + imageName);

        int width = srcImage.getWidth(null);
        int height = srcImage.getHeight(null);

        BufferedImage compatibleImage =
            getCompatibleImage(width, height, transparency);

        Graphics g = compatibleImage.getGraphics();
        g.drawImage(srcImage, 0, 0, width, height, null);
        g.dispose();

        return compatibleImage;
    }

    /**
     * Returns a compatible image with the specified width, heigh
     * and transparency.
     * @param width   Image width
     * @param height  Image height
     * @param transparency  Image transparency
     * @return compatible image with the specified width, heigh
     * and transparency.
     */
    public BufferedImage getCompatibleImage(int width, int height,
            int transparency) {

        return GraphicsHelper.getCompatibleImage(gameFrame,
                width, height, transparency);

    }
    /**
     * Displays or hides the mouse cursor.
     * @param show  True to show the cursor false to hide it.
     */
    public void showCursor(boolean show) {

        String cursorName = show ? GameConstants.GAME_CURSOR : "";

        gameFrame.setCursor(ResourceManager.getCursor(cursorName));
    }

}
```

```java
package game;

import game.graphic.GraphicsHelper;
import game.ship.PlayerShip;

import java.awt.*;

/**
 * The <code>StaticObjectsManager</code> handles the static objects
 * that needs to be rendered on the screen (except for the game menus
 * which the <code>GUIManager</code> handles>
 */
public class StaticObjectsManager {

    private GameLoop gameLoop;

    private Image bgImage;  // Background image
    private final static String defBGImageName = "bg2_1024.jpg";

    /**
     * Construct new StaticObjectsManager and load the
     * default background image.
     * @param gameLoop Reference to the game loop.
     */
    public StaticObjectsManager(GameLoop gameLoop) {

        this.gameLoop = gameLoop;

        // Load the default background image
        setBackgroundImage(defBGImageName);

    }

    /**
     * Sets the background image.
     * @param imageName Name of the image to load (from the images
     * directory).
     */
    public void setBackgroundImage(String imageName) {
        bgImage = gameLoop.getScreenManager().getCompatibleImage(
                imageName, Transparency.OPAQUE);
    }

    /**
     * Render static objects on the screen (background image,
     * player score, etc.).
     */
    public void render(Graphics g) {

        Dimension screenDimention =
            gameLoop.getScreenManager().getScreenDimension();

        // Draw background image
        g.drawImage(bgImage, 0, 0,
                screenDimention.width, screenDimention.height, null);

        // Draw player statistics
        PlayerShip player1Ship =
            gameLoop.getPlayerManager().getPlayer1Ship();
        PlayerShip player2Ship =
            gameLoop.getPlayerManager().getPlayer2Ship();


        g.setFont(new Font(null, Font.BOLD, 12));
        g.setColor(Color.GREEN);

        GraphicsHelper.setAntialiasedText((Graphics2D)g);

        int level = gameLoop.getLevelsManager().getCurrentLevel();
        g.drawString("Level " + level, 10, 15);

        if (player1Ship != null) {
            g.drawString("SCORE: " + player1Ship.getScore(), 10,
screenDimention.height-35);
            g.drawString("POWER: " + player1Ship.getArmor(), 10,
screenDimention.height-20);
        }

        if (player2Ship != null) {

            String scoreText = "SCORE: " + player2Ship.getScore();
            int textWidth = g.getFontMetrics().stringWidth(scoreText);
            int rightAlignment = Math.max(90, textWidth + 10);

            g.drawString(scoreText,
                    screenDimention.width - rightAlignment,
screenDimention.height-35);
            g.drawString("POWER: " + player2Ship.getArmor(),
                    screenDimention.width - rightAlignment,
screenDimention.height-20);
        }
    }

}
```

```java
package game;

import game.network.client.GameNetworkManager;
import game.network.packet.*;
import game.ship.*;
import game.ship.bonus.*;
import game.ship.weapon.*;

import java.awt.*;
import java.util.*;

/**
 * The <code>EnemyShipsManager</code> manage the enemy ships and
 * their shots and bonuses. It also implements the <code>ShipContainer</code>
 * interface to allow the ships to communicate with it.
 */
public class EnemyShipsManager implements Renderable,
    ShipContainer, PacketHandler {

    private final int handlerID = GameConstants.ENEMY_MANAGER_ID;

    private GameLoop gameLoop;

    /** Map of the enemy ships. Object id as key, Ship object as value */
    private Map enemyShips;

    /** Collection of active shots fired by the ships */
    private Collection shots;
    /** Collection of active bonuses dropped by the ships */
    private Collection bonuses;
    /** Collection of targets for the enemy ships (i.e., player ship(s)) */
    private Collection targets;


    /**
     * Construct the EnemyShipsManager, init the collections.
     * @param gameLoop  Reference to the game loop
     */
    public EnemyShipsManager(GameLoop gameLoop) {
        this.gameLoop = gameLoop;
        enemyShips = new HashMap();
        shots = new ArrayList();
        bonuses = new ArrayList();
        targets = new ArrayList();
    }

    /**
     * Get ready for a new level. Clear reminders from previous
     * level and set the enemy ships.
     * @param enemyShips  Map of enemy ships for the current level
     */
    public void newLevel(Map enemyShips) {
        // Make sure no objects left from previous level
        this.shots.clear();
        this.bonuses.clear();
        this.enemyShips.clear();
        setEnemyShips(enemyShips);
    }

    /**
```

```
     * Adds a new target to the targets collection.
     * @param target  New target to add.
     */
    public void addTarget(Target target) {
        targets.add(target);
    }

    /**
     * Adds a collection of targets.
     * @param targets Collection of Target objects.
     */
    public void addTarget(Collection targets) {
        targets.addAll(targets);
    }

    /**
     * Adds ship to the manager and sets its manager to be this object.
     * @param ship Ship to add.
     */
    public void addShip(Ship ship) {
        ship.setShipContainer(this);
        enemyShips.put(new Integer(ship.getHandlerId()), ship);
    }

    /**
     * Adds bonus to the bonuses collection.
     * @param bonus Bonus to add.
     */
    public void addBonus(Bonus bonus) {
        bonuses.add(bonus);
    }

    /**
     * Adds shot to the shots collection.
     * @param shot Shot to add.
     */
    public void addShot(Bullet shot) {
        shots.add(shot);
    }

    /**
     * Sets the enemy ships managed by this object. Sets this object
     * to be the ships container.
     * @param enemyShips  New map of ships.
     */
    private void setEnemyShips(Map enemyShips) {
        Iterator shipsItr = enemyShips.values().iterator();
        while (shipsItr.hasNext()) {
            Ship ship = (Ship) shipsItr.next();
            ship.setShipContainer(this);
        }
        this.enemyShips = enemyShips;
    }

    /**
     * Updates the state of all the managed objects.
     * @param elapsedTime Time elapsed since last call to this method
     * in milliseconds.
     */
    public void update(long elapsedTime){
```

```java
        Dimension screenDimension =
            gameLoop.getScreenManager().getScreenDimension();

        Insets insets =
            gameLoop.getScreenManager().getScreenInsets();

        // Update ships
        Iterator shipsItr = enemyShips.values().iterator();
        while (shipsItr.hasNext()) {
            Ship ship = (Ship) shipsItr.next();

            if (ship.isDestroyed()) {
                // Remove the destroyed ship
        shipsItr.remove();
            }
            else {

              ship.update(elapsedTime);

              // If the ship exits the screen and still in the wrong
              // direction, change its velocity so it will get back
              if (ship.getX() < insets.left && ship.getDx() < 0) {
                  ship.setDx(-ship.getDx());
              }
              if (ship.getX()+ ship.getWidth() >
                      screenDimension.width - insets.right
                      && ship.getDx() > 0) {
                  ship.setDx(-ship.getDx());
              }
              if (ship.getY() < insets.top && ship.getDy() < 0) {
                  ship.setDy(-ship.getDy());
              }
              if (ship.getY() + ship.getHeight() >
                      screenDimension.height - insets.bottom
                      && ship.getDy() > 0) {
                  ship.setDy(-ship.getDy());
              }

            }
        }

        // Update shots
        Iterator shotsItr = shots.iterator();
        while (shotsItr.hasNext()) {
            Bullet shot = (Bullet) shotsItr.next();
            shot.updatePosition(elapsedTime);
            if (isOutOfScreen(shot)) {
                shotsItr.remove();
            }
        }

        // Update bonuses
        Iterator bonusesItr = bonuses.iterator();
        while (bonusesItr.hasNext()) {
            Bonus bonus = (Bonus) bonusesItr.next();
            bonus.updatePosition(elapsedTime);
            if (isOutOfScreen(bonus)) {
                bonusesItr.remove();
            }
        }
        ////////////////////////
        // Process Collisions //
        ////////////////////////

        // Process ship-to-ship collisions
        shipsItr = enemyShips.values().iterator();
        while (shipsItr.hasNext()) {
            Ship ship = (Ship) shipsItr.next();
            ship.processCollisions(targets);
        }

        // Process shots to player ship(s) collisions
        shotsItr = shots.iterator();
        while (shotsItr.hasNext()) {
            Bullet shot = (Bullet) shotsItr.next();
            shot.processCollisions(targets);
            if (shot.isHit()) {
                shotsItr.remove();
            }

        }

        // Process bonuses to player ship(s) collisions
        bonusesItr = bonuses.iterator();
        while (bonusesItr.hasNext()) {
            Bonus bonus = (Bonus) bonusesItr.next();
            bonus.processCollisions(targets);
            if (bonus.isHit()) {
                bonusesItr.remove();
            }
        }

    }

    /**
     * Renders all relevant objects and data (ships, shots, etc.)
     */
    public void render(Graphics g) {
        // Render ships
        Iterator itr = enemyShips.values().iterator();
        while (itr.hasNext()) {
            Sprite ship = (Sprite) itr.next();
            ship.render(g);
        }

        // Render shots
        Iterator shotsItr = shots.iterator();
        while (shotsItr.hasNext()) {
            Sprite shot = (Sprite) shotsItr.next();
            shot.render(g);
        }

        Iterator bonusesItr = bonuses.iterator();
        while (bonusesItr.hasNext()) {
            Sprite bonus = (Sprite) bonusesItr.next();
            bonus.render(g);
        }

    }

    /**
```

```
     * Return true if the sprite is off the screen bounds.
     * @param sprite  Sptite to test.
     * @return  True if the sprite is off the screen bounds
     */
    private boolean isOutOfScreen(Sprite sprite) {
        Dimension screenDimension =
            gameLoop.getScreenManager().getScreenDimension();

        return sprite.getX() + sprite.getWidth() < 0 ||
          sprite.getX() > screenDimension.width ||
          sprite.getY() + sprite.getHeight() < 0 ||
          sprite.getY() > screenDimension.height;
    }

    /**
     * Returns true if the level is finished. The level is finished if
     * all the enemy ships destroyed and there are no active bonuses.
     * @return  True if the level is finished.
     */
    public boolean isLevelFinished() {
        return enemyShips.isEmpty() && bonuses.isEmpty();
    }

    /**
     * Returns true if this macine is the controller.
     */
    public boolean isController() {
        return gameLoop.isController();
    }

    /**
     * Return true if this is a network game.
     */
    public boolean isNetworkGame() {
        return gameLoop.isNetworkGame();
    }

    /**
     * Returns the game network manager. Null if this is not
     * a network game.
     */
    public GameNetworkManager getNetworkManager() {
        return gameLoop.getGameNetworkManager();
    }

    /**
     * Handles incoming packets. If the packet should be handle by
     * the maneger than handle it otherwise search for a ship to
     * to handle it.
     * @param packet Packet to handle
     */
    public void handlePacket(Packet packet) {

        if (packet instanceof BulletPacket) {
            BulletPacket bulletPacket = (BulletPacket)packet;

            BulletModel model = bulletPacket.getBulletModel();

            Ship owningShip = (Ship) enemyShips.get(
                    new Integer(packet.handlerId));
            if (owningShip != null) { // The ship might be destroyed

                Bullet bullet = WeaponFactory.getBullet(model,
                        owningShip);

                addShot(bullet);
            }

            packet.setConsumed(true);

        }
        else if (packet instanceof PowerUpPacket) {
            PowerUpPacket powerPacket = (PowerUpPacket)packet;
            Bonus powerUp = new PowerUp(powerPacket.x,
                    powerPacket.y, powerPacket.powerUp);

            addBonus(powerUp);
            packet.setConsumed(true);
        }
        else if (packet instanceof WeaponUpgradePacket) {
            WeaponUpgradePacket wuPacket = (WeaponUpgradePacket)packet;
            Bonus weaponUpgrade = new WeaponUpgrade(wuPacket.x,
                    wuPacket.y, wuPacket.weaponType);

            addBonus(weaponUpgrade);
            packet.setConsumed(true);
        }
        else {
            // Check if one of the ships can handle it
            Integer handlerID = new Integer(packet.handlerId);
            Ship ship = (Ship) enemyShips.get(handlerID);
            if (ship != null) {
                ship.handlePacket(packet);
            }
        }

    }

    /**
     * Currently this object doesn't creates it's own packets
     */
    public void createPacket(GameNetworkManager netManager) {
        // Currently this object doesn't creates it's own packets
    }

    /**
     * Returns the network handler id of this object.
     */
    public int getHandlerId() {
        return this.handlerID;
    }

}
```

```java
package game;

/**
 * The <code>GameConstants</code> interface holds some game-wide
 * constants.
 */
public final class GameConstants {

    public final static long FRAME_SLEEP_TIME = 20;

    public final static String RESOURCES = "resources";

    public final static String IMAGES = "images";

    public final static String SOUNDS = "sounds";

    public final static String IMAGES_DIR = RESOURCES + "/" + IMAGES + "/";

    public final static String SOUNDS_DIR = RESOURCES + "/" + SOUNDS + "/";

    public final static String CONFIG_DIR = "config/";

    public final static String DBName = "java:comp/env/jdbc/gameDB";

    public final static int PLAYER1_ID = 1;

    public final static int PLAYER2_ID = 2;

    public final static int ENEMY_MANAGER_ID = 3;

    public final static int PLAYER_MANAGER_ID = 4;

    public final static int FIRST_ENEMY_SHIP_ID = 1001;

    public final static String GAME_FONT = "gameFont.ttf";

    public final static String GAME_CURSOR = "targetCur.gif";

}
```

```java
package game;

import game.gamestate.*;
import game.highscore.HighScoresManager;
import game.input.InputManager;
import game.network.client.GameNetworkManager;

import java.util.*;

/**
 * The <code>GameLoop</code> is the object that runs the various
 * game states and switches from one game state to another when it's
 * finished.
 * This class initializes most of the game manager objects and gives
 * the various states the ability to access the managers.
 */
public class GameLoop implements Runnable {

    private ScreenManager screenManager;
    private GUIManager guiManager;
    private LevelsManager levelsManager;
    private InputManager inputManager;
    private GameNetworkManager gameNetworkManager;
    private StaticObjectsManager staticObjectsManager;
    private EnemyShipsManager enemyShipsManager;
    private PlayerManager playerManager;
    private HighScoresManager highScoresManager;
    private boolean networkGame;

    // True if objects random events are controlled from local machine
    private boolean controller;

    // The various game states
    private GameState curGameState, loadingState, runningState,
      addHighScoreState;

    private Map gameStatesById;

    /**
     * Construct the game loop and initialize objects.
     * @param networkGame True if it's a network game.
     * @param controller  True if this machine is the controller.
     * @param highScoresManager The high scores manager
     * @param gnm     Game network manager. Null in a
     * single player game.
     */
    public GameLoop(boolean networkGame, boolean controller,
            HighScoresManager highScoresManager, GameNetworkManager gnm) {

        this.networkGame = networkGame;
        this.controller = controller;
        this.highScoresManager = highScoresManager;
        this.gameNetworkManager = gnm;
        init();
    }

    /**
     * initialize the game managers.
     */
    private void init() {
        screenManager = ScreenManager.getInstance();
        screenManager.setFullScreen();
        guiManager = new GUIManager(this, screenManager);
        inputManager = new InputManager(this/*,
playerManager.getLocalPlayerShip()*/);

        playerManager = new PlayerManager(this);

        screenManager.getFullScreenWindow().addKeyListener(inputManager);

        enemyShipsManager = new EnemyShipsManager(this);
        enemyShipsManager.addTarget(playerManager.getLocalPlayerShip());
        if (networkGame) {
            enemyShipsManager.addTarget(playerManager.getNetworkPlayerShip());
        }

        staticObjectsManager = new StaticObjectsManager(this);

        levelsManager = new LevelsManager(this);

        // Create the various game states and add them to the game
        // states list
        gameStatesById = new HashMap();
        loadingState = new LoadingLevelState(this);
        gameStatesById.put(new Integer(loadingState.getGameStateId()),
                loadingState);

        runningState = new GameRunningState(this);
        gameStatesById.put(new Integer(runningState.getGameStateId()),
                runningState);

        addHighScoreState = new AddHighScoreState(this);
        gameStatesById.put(new Integer(addHighScoreState.getGameStateId()),
                addHighScoreState);

        // Init all the game states
        Iterator gameStatesItr = gameStatesById.values().iterator();
        while(gameStatesItr.hasNext()) {
            GameState gameState = (GameState)gameStatesItr.next();
            gameState.init();
        }

        // Set current game state to loading state
        curGameState = loadingState;

    }

    /**
     * The main loop iterates while the game is not finished and
     * calls the current state methods.
     */
    public void run() {

        long prevFrameTime = System.currentTimeMillis();
        curGameState.start();

        while(!inputManager.isQuit()) {

            long currFrameTime = System.currentTimeMillis();
            try {
                Thread.sleep(GameConstants.FRAME_SLEEP_TIME);
            }
            catch (InterruptedException ie) {
                //
            }

            long elapsedTime = currFrameTime - prevFrameTime;
            prevFrameTime = currFrameTime;

            curGameState.gatherInput(this, elapsedTime);
            curGameState.update(this, elapsedTime);
            curGameState.render(this);

            if (curGameState.isFinished()) {
                changeGameState();
            }

        }

        finalizeGame();

    } // end method run


    /**
     * Finalize the running game. Release resources.
     */
    private void finalizeGame() {
        if (isNetworkGame()) {
            gameNetworkManager.cleanup();
        }
        screenManager.exitFullScreen();
        guiManager.restoreRepaintManager();
    }

  /**
   * Change the current game state.
   * Take the next state from the finished state and
   * call it's start method.
   */
    private void changeGameState() {
        // Switch game state
        int nextStateId = curGameState.getNextGameState();
        curGameState = (GameState)
        gameStatesById.get(new Integer(nextStateId));
        curGameState.start();
    }

    /**
     * Sets this machine to be the controller (if the controller
     * player quits the network game).
     */
    public void setController(boolean controller) {
        this.controller = controller;
    }

    /**
     * Returns true if this machine is the controller.
     */
    public boolean isController() {
        return this.controller;
    }

    /**
     * Sets the networkGame flag. (If the network player quits
     * we turn the flag off to stop sending packets).
     */
    public void setNetworkGame(boolean networkGame) {
        this.networkGame = networkGame;
    }

    /**
     * Returns true if it's a network game.
     */
    public boolean isNetworkGame() {
        return this.networkGame;
    }

    /**
     * Returns the game network manager.
     */
    public GameNetworkManager getGameNetworkManager() {
        return this.gameNetworkManager;
    }

    /**
     * Returns the game network manager.
     */
    public ScreenManager getScreenManager() {
        return screenManager;
    }

    /**
     * Returns the game network manager.
     */
    public PlayerManager getPlayerManager() {
        return playerManager;
    }

    /**
     * Returns the game network manager.
     */
    public LevelsManager getLevelsManager() {
        return this.levelsManager;
    }

    /**
     * Returns the enemy ships manager.
     */
    public EnemyShipsManager getEnemyShipsManager() {
        return this.enemyShipsManager;
    }

    /**
     * Returns the static objects manager.
     */
    public StaticObjectsManager getStaticObjectsManager() {
        return this.staticObjectsManager;
    }

    /**
```

```java
     * Returns the GUI manager.
     */
    public GUIManager getGUIManager() {
        return this.guiManager;
    }

    /**
     * Returns the high scores manager.
     */
    public HighScoresManager getHighScoresManager() {
        return this.highScoresManager;
    }

    /**
     * Returns the input manager.
     */
    public InputManager getInputManager() {
        return this.inputManager;
    }

}
```

```java
package game.gamestate;

import java.awt.Graphics;

import game.GUIManager;
import game.GameLoop;
import game.gui.AddHighScoreDialog;
import game.gui.PostHighScoreDialog;
import game.highscore.HighScore;
import game.highscore.HighScoresManager;
import game.network.packet.Packet;

/**
 * The <code>AddHighScoreState</code> is the last state before the control
 * returns to the game menu. In this state we check if the player scores
 * another high score and ask the player if she wants to post her score.
 */
public class AddHighScoreState implements GameState {

    private final static int INTERNAL_STATE_ADD_HIGH_SCORE = 1;
    private final static int INTERNAL_STATE_POST_SCORE = 2;

    private GameLoop gameLoop;
    private AddHighScoreDialog addHighScoreDialog;
    private PostHighScoreDialog postScoreDialog;
    private HighScoresManager highScoresManager;
    private GUIManager guiManager;
    private String playerName;
    private long timeInState;
    private boolean levelLoaded;
    private int nextGameState;
    private int internalState;
    private long playerScore;
    private int level;
    private boolean finished;

    /**
     * Construct the game state.
     * @param gameLoop  Reference to the game loop.
     */
    public AddHighScoreState(GameLoop gameLoop) {
        this.gameLoop = gameLoop;
        this.highScoresManager = gameLoop.getHighScoresManager();
        this.guiManager = gameLoop.getGUIManager();
    }

    /**
     * Initialize state; create the state dialogs.
     */
    public void init() {

        addHighScoreDialog = new AddHighScoreDialog(gameLoop, this,
                highScoresManager);

        guiManager.addDialog(addHighScoreDialog);

        postScoreDialog = new PostHighScoreDialog(gameLoop,
                highScoresManager);

        guiManager.addDialog(postScoreDialog);
    }
    /**
     * This method is called once when this state is set to be
     * the active state.
     * @see game.gamestate.GameState init method
     */
    public void start() {
        timeInState = 0;
        gameLoop.getScreenManager().showCursor(true);
        finished = false;

        playerScore =
            gameLoop.getPlayerManager().getLocalPlayerShip().getScore();

        level = gameLoop.getLevelsManager().getCurrentLevel();

        if (highScoresManager.isHighScore(playerScore, level)) {

            internalState = INTERNAL_STATE_ADD_HIGH_SCORE;

            popAddHighScoreDialog(playerScore, level);
        }
        else {
            internalState = INTERNAL_STATE_POST_SCORE;
            popPostScoreDialog(new HighScore(playerName, playerScore, level));
        }
    }

    /**
     * This state is gathering input with Swing components.
     */
    public void gatherInput(GameLoop gameLoop, long elapsedTime) {
        // The input is gathered by swing TextField
    }

    /**
     * Update the state. If this state is finished the game is
     * over and we exit the game loop.
     */
    public void update(GameLoop gameLoop, long elapsedTime) {

        timeInState += elapsedTime;

        if (internalState == INTERNAL_STATE_ADD_HIGH_SCORE &&
                addHighScoreDialog.isFinished()) {

            if (playerName != null) {
                internalState = INTERNAL_STATE_POST_SCORE;
                popPostScoreDialog(
                        new HighScore(playerName, playerScore, level));
            }
            else {
                gameLoop.getInputManager().setQuit(true);
            }
        }

        if (internalState == INTERNAL_STATE_POST_SCORE &&
                postScoreDialog.isFinished()) {

            gameLoop.getInputManager().setQuit(true);
```

```java
        }

    }

    /**
     * Render the state data.
     */
    public void render(GameLoop gameLoop) {

        Graphics g = gameLoop.getScreenManager().getGraphics();

        gameLoop.getStaticObjectsManager().render(g);
        gameLoop.getEnemyShipsManager().render(g);
        gameLoop.getGUIManager().render(g);

        g.dispose();

        gameLoop.getScreenManager().show();

    }

    /**
     * No packets are handled in this state.
     */
    public void handlePacket(Packet packet) {
        // No packets to handle in this state
    }

    /**
     * Returns true if this level is finished.
     */
    public boolean isFinished() {
        return finished;
    }

    /**
     * Returns the next game state after this state is finished.
     */
    public int getNextGameState() {
        return nextGameState;
    }

    /**
     * Returns this state id.
     */
    public int getGameStateId() {
        return GameState.GAME_STATE_HIGH_SCORE;
    }

    /**
     * Callback method from the <code>AddHighScoreDialog</code>
     * to store the player name for the next dialog.
     * @param playerName  Name of the player.
     */
    public void setPlayerName(String playerName) {
        this.playerName = playerName;
    }

    /**
     * Show the dialog to add high score.
     * @param score New score to add.
     * @param level Level reached by the player.
     */
    private void popAddHighScoreDialog(long score, int level) {
        addHighScoreDialog.addHighScore(score, level);
    }

    /**
     * Show the dialog for posting a high score.
     * @param score HighScore object to send to the server.
     */
    private void popPostScoreDialog(HighScore score) {
        postScoreDialog.popPostHighScore(score);
    }

}
```

```java
package game.gamestate;

import java.awt.Graphics;

import game.*;
import game.highscore.HighScoresManager;
import game.input.InputManager;
import game.network.client.GameNetworkManager;
import game.network.packet.Packet;

/**
 * The <code>GameRunningState</code> is the central game state. In this
 * state the game is in running mode, meaning the player is able to play
 * and game logic is running through the various manager objects.
 */
public class GameRunningState implements GameState {

    private final static int INTERNAL_STATE_NORMAL = 1;
    private final static int INTERNAL_STATE_LEVEL_CLEARED = 2;

    private int nextGameState;
    private ScreenManager screenManager;
    private GUIManager guiManager;
    private InputManager inputManager;
    private GameNetworkManager gameNetworkManager;
    private StaticObjectsManager staticObjectsManager;
    private EnemyShipsManager enemyShipsManager;
    private PlayerManager playerManager;
    private HighScoresManager highScoresManager;
    private boolean networkGame;
    private boolean finished;
    private int internalState;
    private long timeInState;

    /**
     * Construct the game state.
     * @param gameLoop  Reference to the game loop.
     */
    public GameRunningState(GameLoop gameLoop) {
        this.screenManager = gameLoop.getScreenManager();
        this.guiManager = gameLoop.getGUIManager();
        this.inputManager = gameLoop.getInputManager();
        this.gameNetworkManager = gameLoop.getGameNetworkManager();
        this.staticObjectsManager = gameLoop.getStaticObjectsManager();
        this.enemyShipsManager = gameLoop.getEnemyShipsManager();
        this.playerManager = gameLoop.getPlayerManager();
        this.highScoresManager = gameLoop.getHighScoresManager();
        this.networkGame = gameLoop.isNetworkGame();
    }

    /**
     * Initialize state.
     */
    public void init() {
        // Nothing to initialize
    }

    /**
     * This method is called once when this state is set to be
     * the active state.
     * @see game.gamestate.GameState init method
     */
    public void start() {
        finished = false;
        screenManager.showCursor(false);
        setInternalState(INTERNAL_STATE_NORMAL);
    }

    /**
     * Gather input from the player and from the network.
     */
    public void gatherInput(GameLoop gameLoop, long elapsedTime) {

        inputManager.gatherInput();
        if (!inputManager.isPaused()) {
            playerManager.gatherInput();
        }
        if (networkGame) {
            gameNetworkManager.gatherInput(this);
        }

    }

    /**
     * Update the game state.
     * Most of the game logic starts from here. We call the ships
     * managers update methods.
     */
    public void update(GameLoop gameLoop, long elapsedTime) {
        if (!inputManager.isPaused()) {
            timeInState += elapsedTime;
            enemyShipsManager.update(elapsedTime);
            playerManager.update(elapsedTime);
        }

        if (!(internalState == INTERNAL_STATE_LEVEL_CLEARED) &&
                enemyShipsManager.isLevelFinished()) {
            setInternalState(INTERNAL_STATE_LEVEL_CLEARED);
        }
        else if (playerManager.isGameOver()) {
            finished = true;
            inputManager.setPaused(true);

            nextGameState = GAME_STATE_HIGH_SCORE;

        } else if (internalState == INTERNAL_STATE_LEVEL_CLEARED &&
                timeInState > 2000) {
            finished = true;
            nextGameState = GAME_STATE_LOADING;
        }

    }

    /**
     * Render the game state.
     */
    public void render(GameLoop gameLoop) {
        Graphics g = screenManager.getGraphics();

        staticObjectsManager.render(g);
        enemyShipsManager.render(g);
        playerManager.render(g);
        guiManager.render(g);

        g.dispose();

        screenManager.show();

    }

    /**
     * Handle network packet. This callback method is called
     * from the <code>GameNetworkManager</code> in the input
     * gathering part.
     */
    public void handlePacket(Packet packet) {
        enemyShipsManager.handlePacket(packet);
        playerManager.handlePacket(packet);
    }

    /**
     * Returns true if the current game state is finished.
     * @return  True if the current state is finished
     */
    public boolean isFinished() {
        return finished;
    }

    /**
     * Returns the next game state after the current game state is finished.
     * @return Next game state.
     */
    public int getNextGameState() {
        return nextGameState;
    }

    /**
     * Returns the id of this game state
     * @return  Id of this game state
     */
    public int getGameStateId() {
        return GameState.GAME_STATE_RUNNING;
    }

    /**
     * Sets the internal state to the new state and sets the time
     * in state to 0.
     * @param state New internal state
     */
    private void setInternalState(int state) {
        this.internalState = state;
        this.timeInState = 0;
    }

}
```

```java
package game.gamestate;

import game.GameLoop;
import game.network.packet.Packet;

/**
 * The <code>GameState</code> interface represents a state in the game,
 * like loading, running etc.
 * Each state encapsulate it's own logic and rendering. The game loop
 * calls the current game state gatherInput, update and render methods
 * in a loop. The current game state marks the next game state when
 * it's finished.
 */
public interface GameState {

    public static final int GAME_STATE_RUNNING = 1;
    public static final int GAME_STATE_LOADING = 2;
    public static final int GAME_STATE_HIGH_SCORE = 3;

    /**
     * This method is called once after the <code>GameState</code>
     * object is instantiated. Here all the one time initialization
     * procedures should be performed.
     *
     */
    public void init();

    /**
     * This method is called after a game state changes.
     * Performs any initializations before starting the state.
     */
    public void start();

    /**
     * Gather relevant input (from the keyboard and network)
     * @param gameLoop  Reference to the <code>GameLoop</code> object
     * @param elapsedTime Time elapsed in milliseconds since the last call
     * to this method.
     */
    public void gatherInput(GameLoop gameLoop, long elapsedTime);

    /**
     * Updates the objects and checks for state changes.
     * @param gameLoop  Reference to the <code>GameLoop</code> object
     * @param elapsedTime Time elapsed in milliseconds since the last call
     * to this method.
     */
    public void update(GameLoop gameLoop, long elapsedTime);

    /**
     * Render on the screen
     * @param gameLoop  Reference to the <code>GameLoop</code> object
     */
    public void render(GameLoop gameLoop);

    /**
     * Handles a packet received from the network player. Usually passes
     * the packet to the appropriate game object.
     * @param packet  A packet
     */
    public void handlePacket(Packet packet);
    /**
     * Returns true if the current game state is finished. If so
     * the game loop should switch to the next game state.
     * @return  True if the current state is finished
     */
    public boolean isFinished();

    /**
     * Returns the next game state after the current game state is finished
     * @return Next game state
     */
    public int getNextGameState();

    /**
     * Returns the id of the game state
     * @return  Id of the game state
     */
    public int getGameStateId();


}
```

```java
package game.gamestate;

import game.GameLoop;
import game.LevelsManager;
import game.network.packet.*;
import game.ship.*;
import game.util.Logger;
import game.util.ResourceManager;

import java.awt.*;
import java.util.*;

/**
 * The <code>LoadingLevelState</code> is a game state that responsible
 * to load the next level either from the file (via the <code>LevelsManager
 * </code> or the network).
 * The actual work is done from a seperate thread to allow the state
 * processing input and rendering while the level is loaded.
 */
public class LoadingLevelState implements GameState {

    private GameLoop gameLoop;
    private LevelsManager levelsManager;
    private long timeInState;
    private boolean levelLoaded;
    private boolean friendReady;
    private int nextGameState;
    private String loadingStr1, loadingStr2;
    private boolean finished;
    private boolean gameFinished; // True if we finished the last level

    /**
     * Construct the game state.
     * @param gameLoop  Reference to the game loop.
     */
    public LoadingLevelState(GameLoop gameLoop) {
        this.gameLoop = gameLoop;
        this.levelsManager = gameLoop.getLevelsManager();
    }

    /**
     * Initialize state.
     */
    public void init() {
        // Nothing to initialize
    }

    /**
     * This method is called once when this state is set to be
     * the active state. Start the loader thread.
     * @see game.gamestate.GameState init method
     */
    public void start() {
        timeInState = 0;
        levelLoaded = friendReady = finished = false;

        // The loading thread will change the status text
        loadingStr1 = loadingStr2 = "";

        // Load the new level in a new thread
        new Thread(new LevelLoaderThread()).start();
    }

    /**
     * Check for special input from the player (quit, pause, etc.)
     * Check for network input.
     */
    public void gatherInput(GameLoop gameLoop, long elapsedTime) {

        // Check if the user wants to quit
        gameLoop.getInputManager().gatherInput();

        // Check for network input
        if (gameLoop.isNetworkGame()) {
            gameLoop.getGameNetworkManager().gatherInput(this);;
        }
    }


    /**
     * Update the game state. Check if the level is loaded.
     * If no more levels left display a message and go to the
     * add high score state.
     */
    public void update(GameLoop gameLoop, long elapsedTime) {
        timeInState += elapsedTime;

        if (levelLoaded &&
                ((!gameLoop.isNetworkGame() && timeInState > 300)
                        || friendReady)) {
            finished = true;
            nextGameState = GameState.GAME_STATE_RUNNING;
        }
        else if (gameFinished && timeInState > 7000) {
            // Game finished, display message and goto
            // add high score state
            finished = true;
            nextGameState = GameState.GAME_STATE_HIGH_SCORE;
        }

    }

    /**
     * Render the loading message and dialogs if any.
     */
    public void render(GameLoop gameLoop) {
        Graphics g = gameLoop.getScreenManager().getGraphics();


        gameLoop.getStaticObjectsManager().render(g);
        gameLoop.getPlayerManager().render(g);

        renderLoading(g);

        gameLoop.getGUIManager().render(g);

        gameLoop.getScreenManager().show();
    }

    /**
     * Handle incoming packet.
     * If this is a network game and this computer is not the
     * controller it waits for the controller to send the level details.
     * Otherwise it waits for the ready signal after sending
     * the level details.
     */
    public void handlePacket(Packet packet) {

        if (packet instanceof NewLevelPacket) {
            levelsManager.nextLevel();  // Update the level

            NewLevelPacket newLevel = (NewLevelPacket)packet;
            // Get the enemy ships models
            Collection enemyShipsModels = newLevel.getEnemyShipsModels();

            // Build Ships from the models
            Map enemyShips = new HashMap();
            Iterator modelsItr = enemyShipsModels.iterator();
            while (modelsItr.hasNext()) {
                ShipModel model = (ShipModel) modelsItr.next();
                EnemyShip ship = new EnemyShip(model);
                enemyShips.put(new Integer(ship.getHandlerId()), ship);
            }

            // Update the ship managers
            gameLoop.getEnemyShipsManager().newLevel(enemyShips);
            gameLoop.getPlayerManager().newLevel(enemyShips.values());

            // Signal the network player that this computer is ready to play
            Packet ready = new
SystemPacket(gameLoop.getGameNetworkManager().getSenderId(),
                    gameLoop.getGameNetworkManager().getReceiverId(),
                    SystemPacket.TYPE_READY_TO_PLAY);

            gameLoop.getGameNetworkManager().sendPacket(ready);

            levelLoaded = true;
            friendReady = true;

            packet.setConsumed(true);

        }
        else if (packet instanceof SystemPacket) {
            int type = ((SystemPacket)packet).getType();
            if (type == SystemPacket.TYPE_READY_TO_PLAY) {
                friendReady = true;
            }

            packet.setConsumed(true);
        }
        else {
            // We consume packets anyway in the loading level state
            // to eliminate packets left from the previous level
            packet.setConsumed(true);
        }
    }

    /**
     * Returns true if the current game state is finished.
     * @return  True if the current state is finished
     */
    public boolean isFinished() {
```

```java
        return finished;
    }

    /**
     * Returns the next game state after the current game state is finished.
     * @return Next game state.
     */
    public int getNextGameState() {
        return nextGameState;
    }

    /**
     * Returns the id of this game state
     * @return  Id of this game state
     */
    public int getGameStateId() {
        return GameState.GAME_STATE_LOADING;
    }

    /**
     * Renders loading message on the screen.
     */
    private void renderLoading(Graphics g) {

        // Paint anti-aliased text
        Graphics2D g2d = (Graphics2D)g;
        g2d.setRenderingHint(RenderingHints.KEY_TEXT_ANTIALIASING,
                RenderingHints.VALUE_TEXT_ANTIALIAS_ON);

        Dimension screenDimention =
            gameLoop.getScreenManager().getScreenDimension();

        g.setFont(ResourceManager.getFont(Font.BOLD, 20));
        g.setColor(Color.BLUE);

        FontMetrics metrics = g.getFontMetrics();

        int width = metrics.stringWidth(loadingStr1);
        int middleX = screenDimention.width / 2;
        int middleY = screenDimention.height / 2;

        g.drawString(loadingStr1 , middleX - width/2, middleY);

        width = metrics.stringWidth(loadingStr2);

        g.drawString(loadingStr2 , middleX - width/2,
                middleY + metrics.getHeight() + 5);

    }

    /**
     * Sets the loading level message.
     * @param line1 First line to diplay
     * @param line2 Second line to display
     */
    private void setLoadingStr(String line1, String line2) {
        this.loadingStr1 = line1;
        this.loadingStr2 = line2;
    }

    // Private inner class that implements runnable and used to
    // load new level in a new thread
    private class LevelLoaderThread implements Runnable {

        public void run() {

            Logger.printMemoryUsage("New level memory usage before:");
            // Ask the garbage collector to run before starting a new level
            System.gc();
            Logger.printMemoryUsage("New level memory usage after:");

      if (levelsManager.isLastLevel()) {
          gameFinished = true;
          setLoadingStr("Congratulations!","You have finished the game!");
          return;
      }

      int curLevel = levelsManager.getCurrentLevel();

            setLoadingStr("Loading level " + (curLevel+1) + ".... ", "");

            if (gameLoop.isController()) {
                // Only the controller machine loads the file and then send
                // the data to the other player
                Map enemyShips = levelsManager.loadNextLevel();
                gameLoop.getEnemyShipsManager().newLevel(enemyShips);
                gameLoop.getPlayerManager().newLevel(enemyShips.values());

                // If it's a network game
                if (gameLoop.isNetworkGame()) {

                    // Build ship models collection
                    Collection enemyShipsModels = new
ArrayList(enemyShips.size());
                    Iterator enemyShipsItr = enemyShips.values().iterator();
                    while (enemyShipsItr.hasNext()) {
                        Ship ship = (Ship) enemyShipsItr.next();
                        ShipModel model = ship.getShipModel();
                        enemyShipsModels.add(model);
                    }

                    // Send the level data to the network player
                    Packet newLevel = new NewLevelPacket(
                            gameLoop.getGameNetworkManager().getSenderId(),
                            gameLoop.getGameNetworkManager().getReceiverId(),
                            enemyShipsModels);

                    gameLoop.getGameNetworkManager().sendPacket(newLevel);


                    // Signal ready to play
                    Packet ready = new
SystemPacket(gameLoop.getGameNetworkManager().getSenderId(),
                            gameLoop.getGameNetworkManager().getReceiverId(),
                            SystemPacket.TYPE_READY_TO_PLAY);

                    gameLoop.getGameNetworkManager().sendPacket(ready);

                    setLoadingStr("Level " + (curLevel+1) + " loaded.",
                  "Waiting for online player...");

                }
                levelLoaded = true;

            }
            else if (gameLoop.isNetworkGame()) {
                levelsManager.loadLocalLevelData(curLevel+1);
                setLoadingStr("Loading level " + (curLevel+1) + ".... ",
            "Waiting for online player data...");
            }
        }

    } // end inner class LevelLoaderThread

} // end class LoadingLevelState
```

```java
package game.graphic;

import java.awt.*;
import java.awt.image.BufferedImage;

/**
 * The graphics helper has some static methods to help
 * creating, loading and drawing graphics.
 */
public class GraphicsHelper {

    /**
     * Returns a compatible image for the configuration of the window
     * with the specified width, heigh and transparency.
     * @param window  Window object
     * @param width   Image width
     * @param height  Image height
     * @param transparency  Image transparency
     * @return compatible image with the specified width, heigh
     * and transparency.
     */
    public static BufferedImage getCompatibleImage(Window window,
            int width, int height, int transparency) {

        GraphicsConfiguration gc = window.getGraphicsConfiguration();

        return getCompatibleImage(width, height, transparency, gc);

    }

    /**
     * Returns a compatible image for the default configuration
     * with the specified width, heigh and transparency.
     * @param width   Image width
     * @param height  Image height
     * @param transparency  Image transparency
     * @return compatible image with the specified width, heigh
     * and transparency.
     */
    public static BufferedImage getCompatibleImage(
            int width, int height, int transparency) {

        GraphicsEnvironment ge =
            GraphicsEnvironment.getLocalGraphicsEnvironment();
        GraphicsConfiguration gc =
            ge.getDefaultScreenDevice().getDefaultConfiguration();

        return getCompatibleImage(width, height, transparency, gc);

    }

    /**
     * Returns a compatible image for the graphics configuration
     * with the specified width, heigh and transparency.
     * @param width   Image width
     * @param height  Image height
     * @param transparency  Image transparency
     * @return compatible image with the specified width, heigh
     * and transparency.
     */
    private static BufferedImage getCompatibleImage(int width,
            int height, int transparency, GraphicsConfiguration gc) {

        return gc.createCompatibleImage(width, height, transparency);

    }

    /**
     * Draws the text in the center of the image.
     * @param g     The graphics device
     * @param image Image to draw on
     * @param text  Text to draw
     */
    public static void drawInMiddle(Graphics g, Image image, String text) {

    int width = image.getWidth(null);
    int height = image.getHeight(null);

        FontMetrics fm = g.getFontMetrics();
        int midX = (width - fm.stringWidth(text)) / 2;
        int midY = (height + fm.getHeight()/2) / 2;

        g.drawString(text, midX, midY);

    }

    public static void setAntialiasedText(Graphics2D g) {
        g.setRenderingHint(RenderingHints.KEY_TEXT_ANTIALIASING,
                RenderingHints.VALUE_TEXT_ANTIALIAS_ON);
    }

}
```

```java
package game.gui;

import game.GameConstants;
import game.graphic.GraphicsHelper;
import game.util.ResourceManager;

import java.awt.*;
import java.awt.event.ActionListener;

import javax.swing.*;

/**
 * The <code>GameDialog</cdoe> class extends <code>JDialog</code> and
 * serves as the base class for all the game menu dialogs (
 * the dialogs before starting a game).
 *
 * @see javax.swing.JDialog
 */
public abstract class GameDialog extends JDialog
    implements ActionListener {

    public static final String BTN_BIG_IMAGE = "button_green.png";
    public static final String BTN_SMALL_IMAGE = "btn_small_green.png";

    /**
     * Constructs a ew <code>GameDialog</code>
     * @param owner Owner frame.
     * @param modal True if it's a modal dialog.
     */
    public GameDialog(JFrame owner, boolean modal) {
        // Set owner dialog and set modal
        super(owner, modal);
    getContentPane().setBackground(Color.BLACK);
      setUndecorated(true);
        createGUI();
    }

    /**
     * This method is called from the constructor. In this method
     * the dialog should build its UI.
     */
    protected abstract void createGUI();

    /**
     * Centers the dialog on the screen. This method should be
     * called only after setting the size of the dialog.
     */
    protected void centralizeOnScreen() {
        Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
        Dimension dialogSize = this.getSize();
        this.setLocation(
                Math.max(0,(screenSize.width - dialogSize.width) / 2),
                Math.max(0,(screenSize.height - dialogSize.height) / 2));
    }

    /**
     * Show the dialog on the screen.
     */
    public void popDialog() {
        this.show(true);
    }
    /**
     * Hides the dialog.
     */
    public void hideDialog() {
        this.show(false);
    }

    /**
     * Creates and returns a customized button. Adds the dialog
     * as the button action listener.
     * @param buttonText  Text to display on the button
     * @param toolTipText Button tooltip text
     * @param buttonImage Name of the button image.
     * @return  Customized button.
     */
    public JButton createButton(String buttonText,
            String toolTipText, String buttonImage) {
        JButton button = new JButton();
        button.setToolTipText(toolTipText);

        customizeButton(button, buttonText, buttonImage);

        button.addActionListener(this);

        return button;
    }

    /**
     * Customize the look and text of the button.
     * @param button    JButton to customize
     * @param buttonText  Text for the button
     * @param buttonImage Image for the button
     */
    protected void customizeButton(JButton button, String buttonText,
            String buttonImage) {

        // Load the base image for the button
        Image srcImage = ResourceManager.loadImage(
            GameConstants.IMAGES_DIR + buttonImage);

    int width = srcImage.getWidth(null);
    int height = srcImage.getHeight(null);

    Font menuFont = ResourceManager.getFont(16);

    // Get a compatible translucent image
    Image image = GraphicsHelper.getCompatibleImage(this,
            width, height, Transparency.TRANSLUCENT);
        // Draw the source image and the button text
    // on the tranlucent image with alpha composite of 0.8
    Graphics2D g = (Graphics2D)image.getGraphics();
        Composite alpha = AlphaComposite.getInstance(
            AlphaComposite.SRC_OVER, 0.8f);
        g.setComposite(alpha);
        g.drawImage(srcImage, 0, 0, null);
        g.setFont(menuFont);
        GraphicsHelper.drawInMiddle(g, image, buttonText);
        g.dispose();

        // Create an image icon for the default button image
        ImageIcon iconDefault = new ImageIcon(image);

        // Create a pressed image
        image = GraphicsHelper.getCompatibleImage(this,
            width, height, Transparency.TRANSLUCENT);
        g = (Graphics2D)image.getGraphics();
        alpha = AlphaComposite.getInstance(
                AlphaComposite.SRC_OVER, 0.9f);
        g.setComposite(alpha);
        // a bit lowered and to the right button
        g.drawImage(srcImage, 2, 2, null);
        g.setFont(menuFont);
        GraphicsHelper.drawInMiddle(g, image, buttonText);
        g.dispose();
        ImageIcon iconPressed = new ImageIcon(image);

        // Create disabled button image
        image = GraphicsHelper.getCompatibleImage(this,
            width, height, Transparency.TRANSLUCENT);
        g = (Graphics2D)image.getGraphics();
        alpha = AlphaComposite.getInstance(
                AlphaComposite.SRC_OVER, 0.3f);
        g.setComposite(alpha);
        g.drawImage(srcImage, 0, 0, null);
        g.setFont(menuFont);
        GraphicsHelper.drawInMiddle(g, image, buttonText);
        g.dispose();
        ImageIcon iconDisabled = new ImageIcon(image);

        button.setOpaque(false);
        button.setFocusPainted(false);
        button.setFocusable(false);
        button.setContentAreaFilled(false);
        button.setBorderPainted(false);
        button.setMargin(new Insets(0, 0, 0, 0));
        button.setIcon(iconDefault);
        button.setPressedIcon(iconPressed);
        button.setDisabledIcon(iconDisabled);
        button.setSize(button.getPreferredSize());

    }

    /**
     * Returns a <code>JPanel/<code> with black background.
     * @param lm  Layout manager for the panel.
     */
    protected JPanel createPanel(LayoutManager lm) {
        JPanel panel = new JPanel(lm);
        panel.setBackground(Color.BLACK);
        return panel;
    }

    /**
     * Returns a <code>JLabel</code> with white game font.
     * @param text Text for the label
     */
    protected JLabel createLabel(String text) {
        JLabel label = new JLabel(text);
        label.setForeground(Color.WHITE);
        label.setFont(ResourceManager.getFont(12));
        return label;
    }

    /**
     * Returns <code>JTextField</code> with black background
     * and white foreground.
     */
    protected JTextField createTextField() {
        JTextField textField = new JTextField();
        textField.setBackground(Color.BLACK);
        textField.setForeground(Color.WHITE);
        return textField;
    }

    /**
     * Returns <code>JPasswordField</code> with black background
     * and white foreground.
     */
    protected JPasswordField createPasswordField() {
        JPasswordField passField = new JPasswordField();
        passField.setBackground(Color.BLACK);
        passField.setForeground(Color.WHITE);
        return passField;
    }

}
```

```java
package game.gui;

import game.GameConstants;
import game.ScreenManager;
import game.graphic.GraphicsHelper;
import game.util.ResourceManager;

import java.awt.*;
import java.awt.event.*;

import javax.swing.*;
import javax.swing.border.BevelBorder;

/**
 * The <code>InGameDialog</code> is used as the base class for all
 * the in game dialogs (dialogs when the users plays the game).
 * This class extends <code>JPanel</code> and uses a customized look -
 * image as background, special buttons, border, etc.
 */
public abstract class InGameDialog extends JPanel implements ActionListener {

    public static final String DEFAULT_BG_IMAGE = "indialogbg.jpg";
    public static final String DEFAULT_BTN_IMAGE = "indialogbtn1.png";

    protected ScreenManager screenManager;
    protected Image bgImage;

    /**
     * Construct a new dialog.
     * @param screenManager A screen manager
     * @param imageName   Name of the background image for the dialog.
     */
    public InGameDialog(ScreenManager screenManager, String imageName) {
        this.screenManager = screenManager;

        if (imageName != null && imageName != "") {
            bgImage = ResourceManager.loadImage(
                    GameConstants.IMAGES_DIR + imageName);
        }

        initDialog();
    }

    /**
     * Init the dialog properties.
     */
    protected void initDialog() {

      this.setLayout(new BorderLayout());
    this.setVisible(false);
      this.setOpaque(false);
      this.setBorder(BorderFactory.createBevelBorder(BevelBorder.RAISED));

    }

    /**
     * Creates and returns a customized button.
     * @param buttonText  Text to display on the button
     * @param imageName   Name of the button image.
     * @return  Customized button.
     */
    protected JButton createButton(String buttonText, String imageName) {
        JButton button = new JButton();

        // Load the image for the button
    Image srcImage = ResourceManager.loadImage(
            GameConstants.IMAGES_DIR + imageName);

    int width = srcImage.getWidth(null);
    int height = srcImage.getHeight(null);

    // Get a compatible translucent image
    Image image = screenManager.getCompatibleImage(
            width, height, Transparency.TRANSLUCENT);

        // Draw the source image and the button text
    // on the tranlucent image with alpha composite of 0.5
    Graphics2D g = (Graphics2D)image.getGraphics();
        Composite alpha = AlphaComposite.getInstance(
            AlphaComposite.SRC_OVER, 0.5f);
        g.setComposite(alpha);
        g.drawImage(srcImage, 0, 0, null);
        g.setFont(ResourceManager.getFont(16));
        GraphicsHelper.drawInMiddle(g, image, buttonText);
        g.dispose();

        // Create an image icon for the default button image
        ImageIcon iconDefault = new ImageIcon(image);

        // Create a pressed image
        image = screenManager.getCompatibleImage(
            width, height, Transparency.TRANSLUCENT);
        g = (Graphics2D)image.getGraphics();
        alpha = AlphaComposite.getInstance(
                AlphaComposite.SRC_OVER, 0.7f);
            g.setComposite(alpha);
        g.drawImage(srcImage, 2, 2, null);
        g.setFont(ResourceManager.getFont(16));
        GraphicsHelper.drawInMiddle(g, image, buttonText);
        g.dispose();
        ImageIcon iconPressed = new ImageIcon(image);

        button.setOpaque(false);
        button.setFocusPainted(false);
        button.setFocusable(false);
        button.setContentAreaFilled(false);
        button.setBorderPainted(false);
        button.setMargin(new Insets(0, 0, 0, 0));
        button.setIcon(iconDefault);
        button.setPressedIcon(iconPressed);
        button.setSize(button.getPreferredSize());
        button.addActionListener(this);

        return button;
    }

    /**
     * Center the dialog on the game frame.
     */
    protected void centralizeOnScreen() {
        Dimension screenSize =
            screenManager.getScreenDimension();
```

```java
        Dimension dialogSize = this.getSize();
        this.setLocation(
                Math.max(0,(screenSize.width - dialogSize.width) / 2),
                Math.max(0,(screenSize.height - dialogSize.height) / 2));
    }

    /**
     * Paint the dialog and the background image.
     */
    public void paint(Graphics g) {
        if (bgImage != null) {
            g.drawImage(bgImage, 0, 0, getWidth(), getHeight(), null);
        }
        super.paint(g);
    }

}
```

```java
package game.gui;

import game.GameLoop;
import game.gamestate.AddHighScoreState;
import game.highscore.HighScore;
import game.highscore.HighScoresManager;
import game.util.ResourceManager;

import java.awt.*;
import java.awt.event.ActionEvent;

import javax.swing.*;
import javax.swing.border.Border;

/**
 * The <code>AddHighScoreDialog</code> appears when the game
 * is over and the player made a new (local) high score.
 */
public class AddHighScoreDialog extends InGameDialog {

    private GameLoop gameLoop;
    private HighScoresManager highScoresManager;
    private AddHighScoreState parentState;

    private JTextField nameField;
    private JButton okButton, cancelButton;
    private long score;
    private int level;
    private boolean finished;

    /**
     * Construct the dialog.
     * @param gameLoop  Reference to the game loop.
     * @param parentState Parent <code>GameState</code> (to update
     * the name the user entered).
     * @param highScoresManager The high scores manager.
     */
    public AddHighScoreDialog(GameLoop gameLoop,
            AddHighScoreState parentState,
            HighScoresManager highScoresManager) {

        super(gameLoop.getScreenManager(), DEFAULT_BG_IMAGE);
        this.gameLoop = gameLoop;
        this.highScoresManager = highScoresManager;
        this.parentState = parentState;

        createGUI();

    }

    /**
     * Setup the GUI.
     */
    protected void createGUI() {

        JPanel inputPanel = new JPanel(new FlowLayout());
    inputPanel.setOpaque(false);

        JLabel nameLabel = new JLabel("Enter your name:");
        nameLabel.setFont(ResourceManager.getFont(Font.BOLD, 16));
        inputPanel.add(nameLabel);

        nameField = new JTextField(10);
        nameField.setFont(ResourceManager.getFont(Font.PLAIN, 16));
        inputPanel.add(nameField);

        this.add(inputPanel, BorderLayout.NORTH);

    JPanel buttonsPanel = new JPanel(new GridLayout(1, 2));
    buttonsPanel.setOpaque(false);

    okButton = createButton("OK", DEFAULT_BTN_IMAGE);
    cancelButton = createButton("Cancel", DEFAULT_BTN_IMAGE);

    buttonsPanel.add(okButton);
    buttonsPanel.add(cancelButton);

    this.add(buttonsPanel, BorderLayout.SOUTH);

        Border border = BorderFactory.createTitledBorder(
                "Congratulation! You made a new High Score");

        this.setBorder(border);

        this.setSize(this.getPreferredSize());

        centralizeOnScreen();

    }

    /**
     * Show the dialog and set the score details.
     * @param score Score made by the player.
     * @param level Level reached by the player.
     */
    public void addHighScore(long score, int level) {
        this.finished = false;
        this.score = score;
        this.level = level;
        this.show(true);
        this.requestFocus();
    }

    /**
     * Perform the addition to the high scores table
     * if the player entered the name and clicked ok.
     */
    private void doAdd() {
        String name = nameField.getText();
        if (name.equals("")) {
            return;
        }
        HighScore newHighScore = new HighScore(name, score, level);
        highScoresManager.addScore(newHighScore, true);
        finished = true;
        parentState.setPlayerName(name);
        this.show(false);
    }

    /**
     * Handle user input.
     */
    public void actionPerformed(ActionEvent e) {

        if (e.getSource() == cancelButton) {
            finished = true;
            this.show(false);
        }

        else if (e.getSource() == okButton) {
            doAdd();
        }
        else if (e.getSource() == nameField) {
            doAdd();
        }
    }

    /**
     * Returnd true when the dialog job is done (user clicked
     * to send of closed the dialog).
     * @return  True if this dialog job is done.
     */
    public boolean isFinished() {
        return finished;
    }

}
```

```java
package game.gui;

import game.GameMenu;
import game.network.client.NetworkException;
import game.network.server.ejb.OnlinePlayerModel;
import game.util.Logger;
import game.util.ResourceManager;

import java.awt.*;
import java.awt.event.ActionEvent;
import java.util.*;

import javax.swing.*;
import javax.swing.table.*;

/**
 * The <code>AvailablePlayersDialog</code> is a dialog which displays
 * the available online players and enables the player to invite
 * available player from the list.
 */
public class AvailablePlayersDialog extends GameDialog {

    private GameMenu gameMenu;

    private JButton inviteButton, refreshButton, closeButton;
    private JLabel statusLabel;
    private JTable playersTable;
    private DefaultTableModel tableModel;

    private java.util.List availablePlayers;
    private boolean invitationSent;

    /**
     * Construct the dialog.
     * @param game  Reference to the game menu.
     */
    public AvailablePlayersDialog(GameMenu game) {

        super(game, true);

        this.gameMenu = game;

        this.setTitle("Online Players");

    }

    /**
     * Create the dialog UI.
     */
    protected void createGUI() {

        Container contentPane = this.getContentPane();

    String[] columnsNames =
        new String[]{"Player Name", "Session Start Time"};
    tableModel = new MyTableModel(columnsNames, 0);

    playersTable = new JTable(tableModel);

    playersTable.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
    playersTable.setBackground(Color.BLACK);
    playersTable.setForeground(Color.WHITE);
    playersTable.setFont(ResourceManager.getFont(14));
    playersTable.getTableHeader().setBackground(Color.BLACK);
    playersTable.getTableHeader().setForeground(Color.WHITE);
    playersTable.getTableHeader().setFont(ResourceManager.getFont(16));

    // Resize the table columns
        TableCellRenderer headerRenderer =
            playersTable.getTableHeader().getDefaultRenderer();

        for (int i = 0; i < columnsNames.length; i++) {
            TableColumn column = playersTable.getColumnModel().getColumn(i);

            Component comp = headerRenderer.getTableCellRendererComponent(
              null, column.getHeaderValue(), false, false, 0, 0);

            int headerWidth = comp.getPreferredSize().width;

            comp = playersTable.getDefaultRenderer(tableModel.getColumnClass(i)).
                             getTableCellRendererComponent(
                                     playersTable, columnsNames[i],
                                 false, false, 0, i);
            int cellWidth = comp.getPreferredSize().width;

            column.setPreferredWidth(Math.max(headerWidth, cellWidth));
        }

    contentPane.add(new JScrollPane(playersTable), BorderLayout.CENTER);

        JPanel buttonsPanel = createPanel(new GridLayout(1, 3));

        inviteButton = createButton("Invite",
                "Invite selected player to play", BTN_SMALL_IMAGE);
        buttonsPanel.add(inviteButton);

        refreshButton = createButton("Refresh",
                "Refresh the players list", BTN_SMALL_IMAGE);
        buttonsPanel.add(refreshButton);

        closeButton = createButton("Close",
                "Close the dialog", BTN_SMALL_IMAGE);
        buttonsPanel.add(closeButton);

        JPanel statusPanel = createPanel(new FlowLayout());
        statusLabel = createLabel("Select player to play with");
        statusLabel.setHorizontalAlignment(SwingConstants.LEFT);
        statusLabel.setForeground(new Color(50, 98, 200));
        statusPanel.add(statusLabel);

        JPanel southPanel = createPanel(new GridLayout(2,1));
        southPanel.add(buttonsPanel);
        southPanel.add(statusPanel);

    contentPane.add(southPanel, BorderLayout.SOUTH);

    this.setSize(400, 300);

    centralizeOnScreen();

    }
```

```java
    /**
     * Refresh the online players list and show the dialog.
     */
    public void popDialog() {
        this.refresh();
        super.popDialog();
    }

    /**
     * Hide the dialog.
     */
    public void hideDialog() {
        this.setVisible(false);
    }

    /**
     * Obtains and updates the list of online players.
     */
    private void refresh() {
        try {
          // Clear previous data
          tableModel.setRowCount(0);

          // Get the available players from the server
          availablePlayers =
              gameMenu.getNetworkManager().getAvailablePlayers();

          // Add the players to the table
          Iterator itr = availablePlayers.iterator();
          while (itr.hasNext()) {
              OnlinePlayerModel playerModel = (OnlinePlayerModel) itr.next();

              Date startDate = new Date(playerModel.getSessionStartTime());

              tableModel.addRow(new String[]{playerModel.getUserName(),
                      startDate.toString()});
          }
        }
        catch (NetworkException ne) {
            Logger.exception(ne);
            Logger.showErrorDialog(this, ne.getMessage());
        }

    }

    /**
     * Invite the selected player for online game.
     */
    private void invitePlayer() {
        try {
            // Get the selected row
          int selectedRow = playersTable.getSelectedRow();
          if (selectedRow > -1) {
              // Get the player's session id and send invitation
              OnlinePlayerModel selectedPlayer = (OnlinePlayerModel)
                availablePlayers.get(selectedRow);

              Long sessionId = selectedPlayer.getSessionId();
              gameMenu.getNetworkManager().sendInvitation(sessionId);
              gameMenu.getNetworkManager().acceptInvitations(false);
```

```
                setStatusText("Invitation " +
                    "sent to " + selectedPlayer.getUserName() +
                    ". Waiting for reply...");

                invitationSent = true;
                // Change the invite button to cancel button
                customizeButton(inviteButton, "Cancel", BTN_SMALL_IMAGE);
            }
        }
        catch (NetworkException ne) {
            Logger.exception(ne);
            Logger.showErrorDialog(this, ne.getMessage());
        }
    }

    /**
     * Cancel a previously sent invitation by changing the status
     * and sending cancellation packet.
     */
    public void cancelInvitation() {
        try {
            gameMenu.getNetworkManager().cancelInvitation();
            reset();
        }
        catch (NetworkException ne) {
            Logger.exception(ne);
            Logger.showErrorDialog(this, ne.getMessage());
        }
    }

    /**
     * Reset the invitation sent status and button.
     */
    public void reset() {
        invitationSent = false;
        customizeButton(inviteButton, "Invite", BTN_SMALL_IMAGE);
    }

    /**
     * Sets the status of the invitation.
     * @param text  Status text.
     */
    public void setStatusText(String text) {
        statusLabel.setText(text);
    }

    /**
     * React to the user input.
     */
  public void actionPerformed(ActionEvent event) {

      if (event.getSource() == inviteButton) {
          if (invitationSent) {
              cancelInvitation();
          }
          else {
              invitePlayer();
          }
      }
      else if (event.getSource() == refreshButton) {
          refresh();
      }
      else if (event.getSource() == closeButton) {
          hideDialog();
      }

  }

  /**
   * This inner class is used only to set the editable state
   * of the players to false (the default is true).
   */
  private class MyTableModel extends DefaultTableModel {

      public MyTableModel(String[] columnNames, int rowCount) {
          super(columnNames, rowCount);
      }

      public boolean isCellEditable(int row, int col) {
          return false;
      }
  }
}
```

```java
package game.gui;

import game.highscore.*;
import game.network.client.NetworkException;
import game.util.Logger;

import java.awt.*;
import java.awt.event.ActionEvent;
import java.io.IOException;

import javax.swing.*;

/**
 * The <code>HighScoresDialog<code> displays the game high scores.
 * It can display the local high scores or the high scores from
 * the server. We can also clear the local high scores.
 */
public class HighScoresDialog extends GameDialog {

    private HighScoresManager highScoresManager;
    private HighScore[] highScores;
    private JButton closeButton, clearButton,
      localScoresButton, networkScoresButton;
    private JPanel renderPanel;

    /**
     * Construct the dialog.
     * @param owner Dialog owner frame.
     * @param modal True if it is a modal dialog
     * @param highScoresManager The high scores manager..
     */
    public HighScoresDialog(JFrame owner, boolean modal,
            HighScoresManager highScoresManager) {

        super(owner, modal);

        this.highScoresManager = highScoresManager;
        this.highScores = highScoresManager.getHighScores();

        createGUI();

    }

    /**
     * Create the dialog UI.
     */
    protected void createGUI() {

        setVisible(false);
        setResizable(false);

        Container container = getContentPane();
        container.setLayout(new BorderLayout());

        renderPanel = createPanel(null);
        container.add(renderPanel, BorderLayout.CENTER);

        JPanel buttonsPanel = createPanel(new GridLayout(2,2));

        closeButton = createButton("Close", "", BTN_SMALL_IMAGE);
        buttonsPanel.add(closeButton);
        clearButton = createButton("Clear",
                "Clears the local high scores", BTN_SMALL_IMAGE);
        buttonsPanel.add(clearButton);

        localScoresButton = createButton("Local HS",
                "Display the local high scores", BTN_SMALL_IMAGE);
        buttonsPanel.add(localScoresButton);

        networkScoresButton = createButton("Network HS",
                "Display the high scores from the server", BTN_SMALL_IMAGE);
        networkScoresButton.addActionListener(this);
        buttonsPanel.add(networkScoresButton);

        container.add(buttonsPanel, BorderLayout.SOUTH);

        setSize(350, 400);

        centralizeOnScreen();

    }

    /**
     * Override the paint method to paint the high scores on
     * the render panel.
     */
    public void paint(Graphics g) {
        super.paint(g);

        Graphics rg = renderPanel.getGraphics();

        Rectangle bounds = new Rectangle(0, 0,
                renderPanel.getWidth(), renderPanel.getHeight());

        // Paint high scores on the render panel
        HighScoresRenderer.render(rg, bounds, highScores);

    }

    /**
     * React to user input.
     */
    public void actionPerformed(ActionEvent event) {

        if (event.getSource() == closeButton) {
            setVisible(false);
        }
        else if (event.getSource() == clearButton) {
            highScoresManager.clearHighScores();
            try {
                highScoresManager.saveHighScores();
            } catch (IOException ioe) {
                ioe.printStackTrace();
            }
            repaint();
        }
        else if (event.getSource() == localScoresButton) {
            highScores = highScoresManager.getHighScores();
            repaint();
        }
        else if (event.getSource() == networkScoresButton) {
            getNetworkHighScores();
        }
    }

    /**
     * Gets the high scores from the server and displays them.
     */
    private void getNetworkHighScores() {
        try {
            highScores = highScoresManager.getNetworkHighScores(1, 10);
        }
        catch (NetworkException ne) {
            Logger.showErrorDialog(getOwner(), ne.getMessage());
        }

        repaint();
    }

}
```

```java
package game.gui;

import game.network.client.NetworkException;
import game.network.client.NetworkManager;
import game.network.packet.InvitationPacket;
import game.util.Logger;

import java.awt.*;
import java.awt.event.*;

import javax.swing.*;

/**
 * The <code>InvitationDialog</code> is displayed when an
 * invitation to play arrives from network player.
 * The user can accept or reject the invitation.
 */
public class InvitationDialog extends GameDialog {

    private JButton okButton, cancelButton;
    private JLabel text;
    private NetworkManager networkManager;
    private InvitationPacket invitationPacket;

    /**
     * Construct the dialog.
     * @param owner Owner frame.
     * @param modal True if this is modal dialog.
     * @param networkManager  Network manager.
     */
    public InvitationDialog(JFrame owner, boolean modal,
            NetworkManager networkManager) {

        super(owner, modal);
        this.networkManager = networkManager;

        this.addWindowListener(
                new WindowAdapter() {
                    public void windowClosing(WindowEvent e) {
                        // Cancell the invitation if one exists
                        sendReply(false);
                    }
                }
        );

    }

    /**
     * Create the dialog UI.
     */
    public void createGUI() {

        Container contentPane = this.getContentPane();
        contentPane.setLayout(new BorderLayout());

        text = createLabel("");
        text.setHorizontalAlignment(SwingConstants.CENTER);
        contentPane.add(text, BorderLayout.NORTH);

        JPanel buttonsPanel = createPanel(new GridLayout(1, 2));
        okButton = createButton("OK", "", BTN_SMALL_IMAGE);
        buttonsPanel.add(okButton);

        cancelButton = createButton("Cancel", "", BTN_SMALL_IMAGE);
        buttonsPanel.add(cancelButton);

        contentPane.add(buttonsPanel, BorderLayout.SOUTH);

    this.setSize(300, 150);

    centralizeOnScreen();

    }

    /**
     * Invitation arrived from an online player.
     * @param invitationPacket  Packet with the invitation details.
     */
    public void invitationArrived(InvitationPacket invitationPacket) {

        this.invitationPacket = invitationPacket;
        text.setText("<html>Invitation yo play arrived from " +
                invitationPacket.userName + ".<br>Do you want to accept?" +
                    "</html>");
        super.popDialog();

    }

    /**
     * Previously arrived invitation was cancelled.
     * Display a message.
     */
    public void invitationCancelled() {
        text.setText("Invitation cancelled");
        invitationPacket = null;
        this.validate();
        super.popDialog();
    }

    /**
     * Send reply to an invitation.
     * @param accepted  True if user accepts the invitation. False
     * if the user rejected it or closed the dialog.
     */
    public void sendReply(boolean accepted) {
        if (invitationPacket != null) {
            try {
                networkManager.sendInvitationReply(
                        invitationPacket, accepted);
            }
            catch (NetworkException ne) {
                Logger.exception(ne);
                Logger.showErrorDialog(this, "Unable to send " +
                    "invitation reply: " + ne.getMessage());
            }
        }
        hideDialog();
    }

    /**
```

```java
     * Hides the dialog.
     */
    public void hideDialog() {
        text.setText("");
        super.hideDialog();
    }

    /**
     * Respond to user input.
     */
    public void actionPerformed(ActionEvent event) {

        if (event.getSource() == okButton) {
            sendReply(true);
        }
        else if (event.getSource() == cancelButton) {
            sendReply(false);
        }

    }

}
```

```
package game.gui;

import game.GameMenu;
import game.network.InvalidLoginException;
import game.network.client.NetworkException;

import java.awt.*;
import java.awt.event.ActionEvent;

import javax.swing.*;
import javax.swing.border.BevelBorder;

/**
 * The <code>LoginDialog</code> enables the user to supply a user
 * name and password for logging onto the server.
 */
public class LoginDialog extends GameDialog {

    private GameMenu gameMenu;

    private JTextField userNameField;
    private JPasswordField passwordField;
    private JButton loginButton, cancelButton;
    private JLabel statusLabel;

    /**
     * Construct the dialog.
     * @param game  The game menu.
     */
    public LoginDialog(GameMenu game) {

        super(game, true);

        this.gameMenu = game;
        this.setTitle("Login");
    }

    /**
     * Create the dialog UI.
     */
    protected void createGUI() {

        Container contentPane = this.getContentPane();
        JPanel inputPanel = createPanel(new GridLayout(2, 2));
        inputPanel.add(createLabel(" User Name:"));
        userNameField = createTextField();
    inputPanel.add(userNameField);

    inputPanel.add(createLabel(" Password:"));
    passwordField = createPasswordField();
    passwordField.addActionListener(this);
    inputPanel.add(passwordField);

    contentPane.add(inputPanel, BorderLayout.NORTH);

      loginButton = createButton("Login", "Login", BTN_SMALL_IMAGE);

    cancelButton = createButton("Cancel", "Cancel", BTN_SMALL_IMAGE);

    JPanel buttonsPanel = createPanel(new GridLayout(1, 2));
    buttonsPanel.add(loginButton);
    buttonsPanel.add(cancelButton);

    statusLabel = createLabel("Enter username and password to login");
    statusLabel.setBorder(new BevelBorder(BevelBorder.LOWERED));
    statusLabel.setForeground(Color.RED);

    JPanel southPanel = createPanel(new BorderLayout());
    southPanel.add(buttonsPanel, BorderLayout.NORTH);
    southPanel.add(statusLabel, BorderLayout.SOUTH);

    contentPane.add(southPanel, BorderLayout.SOUTH);

    this.setSize(300, 150);

    centralizeOnScreen();

    }

    /**
     * Login to the server. Get the user name and password from
     * the input fields.
     * If successful update the <code>gameMenu</code> with the logged
     * users details (user name and ticket/session id).
     */
  private void login() {
    String userName = userNameField.getText();
    String password = new String(passwordField.getPassword());
    passwordField.setText("");
    statusLabel.setText("Validating username and password");
    try {
      Long ticket = gameMenu.getNetworkManager().login(userName, password);
      if (ticket == null) {
          throw new InvalidLoginException("Error: null session id received");
      }

      statusLabel.setText(userName + " logged in successfully");

      gameMenu.setLoggedUser(ticket);

      hideDialog();

    }
    catch (InvalidLoginException ile) {
      statusLabel.setText(ile.getMessage());
    }
    catch (NetworkException ne) {
      statusLabel.setText("Error connecting to server");
    }
  }

  /**
   * Hides the dialog and clears the input fields.
   */
  public void hideDialog() {
        // Clear the text and password fields
        userNameField.setText("");
        passwordField.setText("");
      super.hideDialog();
  }

  /**
```

```
     * Respond to user input.
     */
    public void actionPerformed(ActionEvent event) {
        Object source = event.getSource();
        if (source == loginButton || source == passwordField) {
            login();
        }
        else if (event.getSource() == cancelButton) {
            hideDialog();
        }
    }

}
```

```java
package game.gui;

import java.awt.BorderLayout;
import java.awt.Container;
import java.awt.Dimension;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;

import javax.swing.*;


/**
 * The <code>OKDialog</code> is used to display various messages
 * in a blocking dialog with an OK button to close the dialog.
 */
public class OKDialog extends GameDialog {

    private JLabel textLabel;
    private JButton okButton;

    /**
     * Construct the dialog.
     * @param owner Owner frame.
     */
    public OKDialog(JFrame owner) {

        super(owner, true);

        createGUI();

    }

    /**
     * Create the dialog UI.
     */
    public void createGUI() {

        Container contentPane = this.getContentPane();
        contentPane.setLayout(new BorderLayout());

        textLabel = createLabel("");
        textLabel.setHorizontalAlignment(SwingConstants.CENTER);
        contentPane.add(textLabel, BorderLayout.NORTH);

        JPanel buttonsPanel = createPanel(new GridLayout(1, 1));

        okButton = createButton("OK", "", BTN_SMALL_IMAGE);
        buttonsPanel.add(okButton);

        contentPane.add(buttonsPanel, BorderLayout.SOUTH);

    this.pack();

    centralizeOnScreen();
    }

    /**
     * Sets the message this dialog displays
     * @param text  Text to display
     */
    public void setText(String text) {
        textLabel.setText(text);
    }

    /**
     * Hides the dialog and clears the text.
     */
    public void hideDialog() {
        super.hideDialog();
        textLabel.setText("");
    }

    public void popDialog() {
        this.pack();
        super.popDialog();
    }

    public Dimension getMinimumSize() {
        return getPreferredSize();
    }

    public Dimension getPreferredSize() {
        return new Dimension(200, 200);
    }

    /**
     * Clicked on ok button, hide the dialog
     */
    public void actionPerformed(ActionEvent event) {
        hideDialog();
    }

}
```

```java
package game.gui;

import java.awt.*;
import java.awt.event.*;

import game.GameLoop;
import game.highscore.HighScore;
import game.highscore.HighScoresManager;
import game.network.client.NetworkException;
import game.util.ResourceManager;

import javax.swing.*;

/**
 * The <code>PostHighScoreDialog</code> appears when the player
 * finishes a game asking to post the score to the server.
 */
public class PostHighScoreDialog extends InGameDialog {

    private GameLoop gameLoop;
    private HighScoresManager highScoresManager;
    private JTextField nameField;
    private JButton yesButton, noButton;
    private boolean finished;
    private HighScore score;

    /**
     * Construct the dialog.
     * @param gameLoop  Reference to the game loop
     * @param highScoresManager High scores manager object.
     */
    public PostHighScoreDialog(GameLoop gameLoop,
            HighScoresManager highScoresManager) {

        super(gameLoop.getScreenManager(), DEFAULT_BG_IMAGE);
        this.gameLoop = gameLoop;
        this.highScoresManager = highScoresManager;

        createGUI();

    }

    /**
     * Set up the GUI.
     */
    protected void createGUI() {

        JPanel labelPanel = new JPanel();
        labelPanel.setOpaque(false);

        JLabel label = new JLabel("Post your score to the server?");
        label.setFont(ResourceManager.getFont(Font.BOLD, 16));
        labelPanel.add(label);

        JPanel inputPanel = new JPanel(new FlowLayout());
        inputPanel.setOpaque(false);

        JLabel nameLabel = new JLabel("Name:");
        nameLabel.setFont(ResourceManager.getFont(Font.BOLD, 16));
        inputPanel.add(nameLabel);
        nameField = new JTextField(10);
        nameField.setFont(ResourceManager.getFont(Font.PLAIN, 16));
        inputPanel.add(nameField);

        JPanel northPanel = new JPanel(new GridLayout(2, 1));
        northPanel.setOpaque(false);

        northPanel.add(labelPanel);
        northPanel.add(inputPanel);

        this.add(northPanel, BorderLayout.NORTH);

    JPanel buttonsPanel = new JPanel(new GridLayout(1, 2));
    buttonsPanel.setOpaque(false);

    yesButton = createButton("Yes", DEFAULT_BTN_IMAGE);
    noButton = createButton("No", DEFAULT_BTN_IMAGE);

    buttonsPanel.add(yesButton);
    buttonsPanel.add(noButton);

    this.add(buttonsPanel, BorderLayout.SOUTH);

        this.setSize(this.getPreferredSize());

        centralizeOnScreen();

    }

    /**
     * Show the dialog and set the current high score.
     * @param score High score to post.
     */
    public void popPostHighScore(HighScore score) {
        this.score = score;
        nameField.setText(score.getPlayerName());
        this.show(true);
    }

    /**
     * Returnd true when the dialog job is done (user clicked
     * to send of closed the dialog).
     * @return  True if this dialog job is done.
     */
    public boolean isFinished() {
        return finished;
    }

    /**
     * If <code>post</code> is true, send the score to the server and close
     * the dialog. Else just close the dialog.
     * @param post  True if the player chose to post the score.
     */
    private void post(boolean post) {
        if (post) {
          try {
              score.setPlayerName(nameField.getText());
              highScoresManager.postScoreToServer(score);
          }
          catch (NetworkException ne) {
              System.out.println(ne.getMessage());
          }
        }
        this.finished = true;
        this.show(false);
    }

    /**
     * Hanlde the user input.
     */
    public void actionPerformed(ActionEvent e) {

        if (e.getSource() == noButton) {
            post(false);
        }
        else if (e.getSource() == yesButton) {
            if (nameField.getText().equals("")) {
                return;
            }
            post(true);
        }

    }

}
```

```java
package game.gui;

import game.ScreenManager;
import game.input.InputManager;
import game.util.ResourceManager;

import java.awt.*;
import java.awt.event.ActionEvent;

import javax.swing.*;

/**
 * The <code>QuitDialog</code> appears when the user clicks
 * on the quit button to verify she realy wants to quit.
 */
public class QuitDialog extends InGameDialog {

    private InputManager inputManager;

    private JButton yesButton, noButton;

    /**
     * Construct the dialog.
     * @param screenManager Reference to the screen manager.
     * @param inputManager  Input manager.
     */
    public QuitDialog(ScreenManager screenManager,
            InputManager inputManager) {

        super(screenManager, DEFAULT_BG_IMAGE);
        this.inputManager = inputManager;
        createGUI();

    }

    /**
     * Set up the dialog GUI.
     */
    protected void createGUI() {

        JLabel label = new JLabel("Leaving so soon?", SwingConstants.CENTER);
        label.setFont(ResourceManager.getFont(Font.BOLD, 16));

        this.add(label, BorderLayout.CENTER);

    JPanel buttonsPanel = new JPanel(new GridLayout(1, 2));
    buttonsPanel.setOpaque(false);

    yesButton = createButton("Yes", DEFAULT_BTN_IMAGE);
    noButton = createButton("No", DEFAULT_BTN_IMAGE);

    buttonsPanel.add(yesButton);
    buttonsPanel.add(noButton);

    this.add(buttonsPanel, BorderLayout.SOUTH);

        this.setSize(300, 100);

        centralizeOnScreen();

    }
    /**
     * Handle the user input.
     */
    public void actionPerformed(ActionEvent e) {

        if (e.getSource() == noButton) {
            this.show(false);
            screenManager.showCursor(false);
            inputManager.setQuit(false);
        }
        else if (e.getSource() == yesButton) {
            this.show(false);
            screenManager.showCursor(false);
            inputManager.setQuit(true);
        }

    }

}
```

```java
package game.gui;

import game.GameMenu;
import game.network.InvalidLoginException;
import game.network.client.NetworkException;

import java.awt.*;
import java.awt.event.ActionEvent;

import javax.swing.*;
import javax.swing.border.BevelBorder;

/**
 * The <code>SignupDialog</code> enables the user to signup for
 * a new account. The user must supply user name and password.
 */
public class SignupDialog extends GameDialog {

    private GameMenu gameMenu;

    private JTextField userNameField, emailField;
    private JPasswordField passwordField;
    private JButton signupButton, cancelButton;
    private JLabel statusLabel;

    /**
     * Construct the dialog.
     * @param game  The game menu.
     */
    public SignupDialog(GameMenu game) {

        super(game, true);

        this.gameMenu = game;
        this.setTitle("Login");

    }

    /**
     * Create the dialog UI.
     */
    protected void createGUI() {

        Container contentPane = this.getContentPane();

    JPanel inputPanel = createPanel(new GridLayout(3, 2));

    // User name area
    inputPanel.add(createLabel(" User Name:"));
    userNameField = createTextField();
    inputPanel.add(userNameField);

    // Password area
    inputPanel.add(createLabel(" Password:"));
    passwordField = createPasswordField();
    inputPanel.add(passwordField);

    // Email area
    inputPanel.add(createLabel(" Email:"));
    emailField = createTextField();
    inputPanel.add(emailField);
```

```
        contentPane.add(inputPanel, BorderLayout.NORTH);

        // Create buttons
        signupButton = createButton("Signup",
                "Signup for a new account", BTN_SMALL_IMAGE);

        cancelButton = createButton("Cancel", "", BTN_SMALL_IMAGE);

        JPanel buttonsPanel = createPanel(new GridLayout(1, 2));
        buttonsPanel.add(signupButton);
        buttonsPanel.add(cancelButton);

        // Create status label
        statusLabel = createLabel("Enter details to signup");
        statusLabel.setBorder(new BevelBorder(BevelBorder.LOWERED));
        statusLabel.setForeground(Color.RED);

        JPanel southPanel = createPanel(new BorderLayout());
        southPanel.add(buttonsPanel, BorderLayout.NORTH);
        southPanel.add(statusLabel, BorderLayout.SOUTH);

        contentPane.add(southPanel, BorderLayout.SOUTH);

        this.setSize(300, 150);

        centralizeOnScreen();

        }

        /**
         * Respond to user input.
         */
    public void actionPerformed(ActionEvent event) {

        if (event.getSource() == signupButton) {

            signup();

        } else if (event.getSource() == cancelButton) {

            hideDialog();
        }

    }

    /**
     * Clear the input fields and hide the dialog.
     */
    public void hideDialog() {
        // Clear the fields
          userNameField.setText("");
          passwordField.setText("");
          emailField.setText("");
          super.hideDialog();
    }

    /** TODO: handle the case where user already exists here and in the bean */
    /**
     * Sign up for a new account and login afterwards.
     */
```

```java
  private void signup() {

    String userName = userNameField.getText();
    String password = new String(passwordField.getPassword());
    String email = emailField.getText();
    passwordField.setText("");
    statusLabel.setText("Creating new user....");

    try {
      // Register the new user
      gameMenu.getNetworkManager().signup(
                userName, password, email);

      // Logout before trying login with the new user name
      gameMenu.logout();

      // Login with the new user to get session id
      Long ticket = gameMenu.getNetworkManager().login(userName, password);

      if (ticket == null) {
          throw new InvalidLoginException("Error: null session id received");
      }

      statusLabel.setText(userName + " logged in successfully");

      // Set the logged user in the gameMenu
      gameMenu.setLoggedUser(ticket);

      hideDialog();

    }
    catch (InvalidLoginException ile) {
        statusLabel.setText(ile.getMessage());
    }
    catch (NetworkException ne) {
        statusLabel.setText("Error connecting to server");
    }
  }

}
```

```java
package game.highscore;

import java.io.Serializable;

/**
 * The <code>HighScore</code> class encapsulates a player
 * score details.
 */
public class HighScore implements Serializable, Comparable {

    private String playerName;
    private long score;
    private int level;

    /**
     * Construct a new HighScore object.
     * @param playerName  Name of the player
     * @param score     Score the player reached
     * @param level     Level the player reached in the game.
     */
    public HighScore(String playerName, long score, int level) {

        this.playerName = playerName;
        this.score = score;
        this.level = level;

    }

    /**
     * Returns the level reached by the player.
     * @return Level reached by the player.
     */
    public int getLevel() {
        return level;
    }

    /**
     * Sets the level.
     * @param level Level to set.
     */
    public void setLevel(int level) {
        this.level = level;
    }

    /**
     * Returns the player name.
     * @return  Player name.
     */
    public String getPlayerName() {
        return playerName;
    }

    /**
     * Sets the player name.
     * @param playerName  Player name.
     */
    public void setPlayerName(String playerName) {
        this.playerName = playerName;
    }
    /**
     * Returns the player score.
     * @return  The score
     */
    public long getScore() {
        return score;
    }

    /**
     * Sets the score.
     * @param score Score to set.
     */
    public void setScore(long score) {
        this.score = score;
    }

    /**
     * Compare two high scores.
     * @see java.lang.Comparable
     */
    public int compareTo(Object obj) {

        HighScore score = (HighScore)obj;

        if (this.score < score.score) {
            return -1;
        }
        else if (this.score > score.score) {
            return 1;
        }
        else { // (this.score == score.score)
            if (this.level < score.level) {
                return -1;
            }
            else if (this.level > score.level) {
                return 1;
            }
            else {
                return 0;
            }
        }
    }

    public String toString() {
        return playerName + "\t\t" + score + "\t" + level;
    }
}
```

```java
package game.highscore;

import game.GameConstants;
import game.network.client.NetworkException;
import game.network.client.NetworkManager;

import java.io.*;

/**
 * The <code>HighScoresManager</code> class manages the game high
 * scores. It loads and saves the scores to a file.
 * We use this class to post high scores to the game server and
 * to get the best scores from the server.
 */
public class HighScoresManager {

    private final String HIGH_SCORES_FILE_NAME =
        GameConstants.RESOURCES + "/" + "scores.dat";

    private int numOfHighScores;  // Max number of high scores
    private HighScore[] highScores; // Array with all the local high scores

    private NetworkManager networkManager;

    /**
     * Construct new HighScoresManager and load the high scores from a file.
     * @param numOfHighScores Max number of high scores to hold.
     */
    public HighScoresManager(int numOfHighScores) {

        this.numOfHighScores = numOfHighScores;
        this.highScores = new HighScore[numOfHighScores];

        try {
            loadHighScores(false);
        }
        catch (IOException ioe) {
            System.err.println("Unable to read the high scores file");
            ioe.printStackTrace();
        }
        catch (ClassNotFoundException cnfe) {
            System.err.println("Unable to read the high scores file");
            cnfe.printStackTrace();
        }

    }

    /**
     * Sets the network manager for server communication.
     * @param networkManager  The network manager object.
     */
    public void setNetworkManager(NetworkManager networkManager) {
        this.networkManager = networkManager;
    }

    /**
     * Returns array with all the high scores.
     */
    public HighScore[] getHighScores() {
        return this.highScores;
    }
    /**
     * Returns max number of high scores.
     */
    public int getNumOfHighScores() {
        return numOfHighScores;
    }

    /**
     * Adds score to the high scores list. If the score is smaller or equals
     * to the current smallest score it is not added to the list. If the score
     * is not the smallest and the list is full we remove the last score.
     * @param score New high score to insert to the list.
     */
    public void addScore(HighScore score, boolean save) {
        int place = -1;
        boolean placeFound = false;
        for (int i = 0; !placeFound && i < highScores.length; i++) {
            if (highScores[i] == null || score.compareTo(highScores[i]) > 0) {
                placeFound = true;
                place = i;
            }
        }

        if (placeFound) {
            if (highScores[place] == null || place == highScores.length-1) {
                // If null or last place in array simply insert the new
                // high score to the proper place
                highScores[place] = score;
            }
            else {
                // Insert the new high score to the middle of the high scores
                // array. Move the lower scores one place. Remove the last one
                // if high scores exceeds numOfHighScores (the array length).
                for (int i = highScores.length - 1; i > place; i--) {
                    highScores[i] = highScores[i-1];
                }

                // Insert the new high score to the proper place
                highScores[place] = score;
            }

            if (save) {
              // Save the high scores list to file
              try {
                  saveHighScores();
              }
              catch (IOException ioe) {
                  System.err.println("Unable to save the high scores file");
                  ioe.printStackTrace();
              }
            }

        }

    }

    /**
     * Returns true if the input score and level has place in the
     * high scores list.
     */
    public boolean isHighScore(long score, int level) {
        return isHighScore(new HighScore(null, score, level));
    }

    /**
     * Returns true if the input high score has place in the
     * high scores list.
     */
    public boolean isHighScore(HighScore score) {
        if (score.getScore() > 0 &&
                (highScores[numOfHighScores-1] == null ||
                highScores[numOfHighScores-1].compareTo(score) < 0)) {
            return true;
        }

        return false;
    }


    /**
     * Return array of <code>HighScore</code> objects containing the high
     * scores starting at <code>fromPlace</code> inclusive and ending at
     * <code>toPlace</code> inclusive.
     * Place 1 is the highest score.
     * @param fromPlace
     * @param toPlace
     * @return  Array of high scores
     * @throws IllegalArgumentException If fromPlace is smaller then 1
     */
    public HighScore[] getHighScores(int fromPlace, int toPlace)
        throws IllegalArgumentException {

        if (fromPlace < 1 || toPlace > numOfHighScores) {
            throw new IllegalArgumentException("Input out of bounds");
        }

        if (fromPlace > toPlace) {
            throw new IllegalArgumentException("First argument must be " +
                "smaller or equal to the second argument");
        }

        HighScore[] ret = new HighScore[toPlace-fromPlace+1];

        System.arraycopy(highScores, fromPlace-1, ret, 0, ret.length);

        return ret;
    }


    /**
     * Return array of <code>HighScore</code> objects containing the high
     * scores starting at <code>fromPlace</code> inclusive ending at
     * <code>toPlace</code> inclusive.
     * Place 1 is the highest score.
     * @param fromPlace Place of the high score to start with
     * @param toPlace Place of the high score to end with
     * @throws IllegalArgumentException If <code>fromPlace</code> is
     * smaller then 1 or smaller then <code>toPlace</code>
     * @return Array of high scores from the server.
     */
    public HighScore[] getNetworkHighScores(int fromPlace, int toPlace)
```

```
      throws NetworkException, IllegalArgumentException {

        if (fromPlace < 1) {
            throw new IllegalArgumentException("Input out of bounds");
        }

        if (fromPlace < 1 || fromPlace > toPlace) {
            throw new IllegalArgumentException("First argument must be " +
                "smaller or equal to the second argument");
        }

        if (networkManager == null) {
            throw new NetworkException("NetworkManager is null. " +
                "Please restart the game");
        }

        return networkManager.getHighScores(fromPlace, toPlace);

    }

    /**
     * Clears the high scores list.
     */
    public void clearHighScores() {
        for (int i = 0; i < highScores.length; i++) {
            highScores[i] = null;
        }
    }

    /**
     * Loads the high scores from a file.
     * @param clearOldScores  True if we want to clear the high
     * scores array we hold in this object.
     * @throws IOException  Error reading the file.
     * @throws ClassNotFoundException Class of a serialized HighScore
     * cannot be found
     */
    public void loadHighScores(boolean clearOldScores) throws
        IOException, ClassNotFoundException {

        if (clearOldScores) {
            clearHighScores();
        }

        File scoresFile = new File(HIGH_SCORES_FILE_NAME);

        if (scoresFile.exists()) {
            ObjectInputStream ois = new ObjectInputStream(
                    new FileInputStream(scoresFile));

            HighScore score;
            try {
              while ((score = (HighScore)ois.readObject()) != null) {
                  this.addScore(score, false);
              }
            }
            catch (EOFException eofe) {
                // Reached the end of file
            }

            ois.close();
        }

    }

    /**
     * Saves the array of high scores to a file.
     * @throws IOException  Error saving to a file.
     */
    public void saveHighScores() throws IOException {

        File scoresFile = new File(HIGH_SCORES_FILE_NAME);

        if (!scoresFile.exists()) {
            scoresFile.createNewFile();
        }

        ObjectOutputStream oos = new ObjectOutputStream(
                new FileOutputStream(scoresFile));

        for (int i = 0; i < highScores.length && highScores[i] != null; i++) {
            oos.writeObject(highScores[i]);
        }

        oos.close();
    }

    /**
     * Posts a score to the server via the network manager object.
     * @param score Score to post
     * @throws NetworkException If the network manager doesn't exist or
     * error when trying to post the score.
     */
    public void postScoreToServer(HighScore score) throws NetworkException {

        if (networkManager == null) {
            throw new NetworkException("NetworkManager is not initialized");
        }

        networkManager.postHighScore(score);

    }

    public String toString() {

        String ret = "High Scores:\n" +
          "Place\t" + "Name\t\t" + "Score\t" + "Level\n\n";

        for (int i = 0; i < highScores.length && highScores[i] != null; i++) {
            ret += (i+1) + ".\t" + highScores[i] + "\n";
        }

        return ret;
    }

}
```

```java
package game.highscore;

import game.util.ResourceManager;

import java.awt.*;

/**
 * The <code>HighScoresRenderer</code> class is used to render the high
 * scores given a <code>Graphics</code> object with bounding rectangle.
 */
public class HighScoresRenderer {

    /**
     * Render the input high scores using the gtaphic device.
     * @param g Graphics object
     * @param highScores  High scores array
     */
    public static void render(Graphics g, Rectangle bounds,
            HighScore[] highScores) {

        if (highScores == null) {
            // Create empty high scores array
            highScores = new HighScore[1];
        }

        // Get the number of occupied high scores
        int numberOfHighScores = 0;
        if (highScores[0] == null) {
            numberOfHighScores = 0;
        } else {
            int i;
            for (i = 0; i < highScores.length && highScores[i] != null; i++) {
                // iterate
            }
            numberOfHighScores = i;
        }

        final int numColumns = 4;
        final int leftMargins = 10; // 10 pixels from the left
        final int topMargins = 20;  // 20 pixels from the top

        int columnWidth = bounds.width / numColumns;

        // Each column takes different percentage of the screen width
        // Rank column is the narrowest and name is the widest
        int rankWidth = (int)Math.round(bounds.width * 0.15f);
        int nameWidth = (int)Math.round(bounds.width * 0.35f);
        int scoreWidth = (int)Math.round(bounds.width * 0.30f);

        // Calculate the columns places
        int rankPlace = leftMargins;
        int namePlace = rankPlace + rankWidth;
        int scorePlace = namePlace + nameWidth;
        int levelPlace = scorePlace + scoreWidth;

        FontMetrics fm = g.getFontMetrics();
        int fontHeight = fm.getHeight();
        int horizontalSpace = fontHeight + 5;

        g.setColor(Color.BLACK);
        g.fillRect(0, 0, bounds.width, bounds.height);

        g.setColor(Color.BLUE);

        g.setFont(ResourceManager.getFont(Font.BOLD, 16));
        // Draw headline
        g.drawString("Rank",  rankPlace,  topMargins);
        g.drawString("Player",  namePlace,  topMargins);
        g.drawString("Score",   scorePlace, topMargins);
        g.drawString("Level",   levelPlace, topMargins);

        g.setFont(ResourceManager.getFont(Font.PLAIN, 14));
        for (int i = 0; i < numberOfHighScores; i++) {

            // Draw the high score rank
            g.drawString((i+1)+"", rankPlace,
                    topMargins + horizontalSpace*(i+1));

            // Draw player name
            String playerName = highScores[i].getPlayerName();
            if (fm.stringWidth(playerName) > columnWidth) {
                // If string too long take only the first 7 chars
                // and add three dots
                playerName = playerName.substring(0, 7) + "...";
            }
            g.drawString(playerName, namePlace,
                    topMargins + horizontalSpace*(i+1));

            // Draw score
            g.drawString(highScores[i].getScore()+"", scorePlace,
                    topMargins + horizontalSpace*(i+1));

            // Draw level
            g.drawString(highScores[i].getLevel()+"", levelPlace,
                    topMargins + horizontalSpace*(i+1));
        }

    }

}
```

```
package game.highscore;

import java.awt.*;
import java.awt.event.*;
import java.io.IOException;

import javax.swing.*;

/**
 * Simple class to test the HighScoresManager.
 * This class is not part of the game.
 */
public class HighScoresTester extends JFrame implements ActionListener {

    private JTextField name, score, level;
    private JTextArea highScoresArea;
    private JButton add, show, save, load;

    private HighScoresManager highScoresManager;

    public static void main(String args[]) {
        HighScoresTester tester = new HighScoresTester();
        tester.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    public HighScoresTester() {

        highScoresManager = new HighScoresManager(4);

        setupGUI();

    }

    private void setupGUI() {

        Container container = getContentPane();
        container.setLayout(new BorderLayout());

        // Create input fields and labels
        JLabel nameLabel = new JLabel("Name:");
        name = new JTextField(10);
        JLabel scoreLabel = new JLabel("Score:");
        score = new JTextField(10);
        JLabel levelLabel = new JLabel("Level:");
        level = new JTextField(10);

        // Create panel for input fields and add filed to panel
        JPanel scorePanel = new JPanel(new FlowLayout());
        scorePanel.add(nameLabel);
        scorePanel.add(name);
        scorePanel.add(scoreLabel);
        scorePanel.add(score);
        scorePanel.add(levelLabel);
        scorePanel.add(level);

        // Create buttons to control high scores
        add = new JButton("Add");
        add.addActionListener(this);

        show = new JButton("Show");
        show.addActionListener(this);

        save = new JButton("Save");
        save.addActionListener(this);

        load = new JButton("Load");
        load.addActionListener(this);

        // Add the buttons to the buttons panel
        JPanel buttonsPanel = new JPanel(new GridLayout(2,2));
        buttonsPanel.add(add);
        buttonsPanel.add(show);
        buttonsPanel.add(save);
        buttonsPanel.add(load);

        highScoresArea = new JTextArea(10, 40);
        highScoresArea.setEditable(false);

        container.add(scorePanel, BorderLayout.NORTH);
        container.add(new JScrollPane(highScoresArea), BorderLayout.CENTER);
        container.add(buttonsPanel, BorderLayout.SOUTH);

        pack();
        show();

    }


    private void addScore() {
        String name = this.name.getText();
        long score = Long.parseLong(this.score.getText());
        int level = Integer.parseInt(this.level.getText());
        HighScore newScore = new HighScore(name, score, level);

        highScoresManager.addScore(newScore, false);

        showHighScores();

    }

    private void showHighScores() {
        highScoresArea.setText(highScoresManager.toString());
    }

    private void save() {
        try {
            highScoresManager.saveHighScores();
        }
        catch (IOException ioe) {
            System.err.println("Unable to save high scores to file");
            ioe.printStackTrace();
        }
    }

    private void load() {
        try {
            highScoresManager.loadHighScores(true);
        }
        catch (IOException ioe) {
            System.err.println("Unable to load high scores from file");
            ioe.printStackTrace();
        }
        catch (ClassNotFoundException cnfe) {
            System.err.println("Unable to load high scores from file");
            cnfe.printStackTrace();
        }
    }

    public void actionPerformed(ActionEvent event) {

        if (event.getSource() == add) {
            addScore();
        }
        else if (event.getSource() == show) {
            showHighScores();
        }
        else if (event.getSource() == save) {
            save();
        }
        else if (event.getSource() == load) {
            load();
        }
    }

}
```

```java
package game.input;

import game.GameLoop;
import game.gui.QuitDialog;

import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.util.Map;
import java.util.HashMap;

/**
 * The input manager handles the player input from the
 * keyboard while the game is running.
 */
public class InputManager implements KeyListener {

    private GameLoop gameLoop;
    private QuitDialog quitDialog;

    // Game actions
    private GameAction exit;
    private GameAction pause;
    public GameAction moveLeft;
    public GameAction moveRight;
    public GameAction moveUp;
    public GameAction moveDown;
    public GameAction fireBullet;
    public GameAction fireMissile;

    private boolean paused = false;
    private boolean quit = false;

    /** Virtual keys to GameAction map */
    private Map keyCodeToGameAction;

    /**
     * Construct the input manager. Create the game actions.
     * @param gameLoop  Reference to the GameLoop
     */
    public InputManager(GameLoop gameLoop) {

        this.gameLoop = gameLoop;

        this.quitDialog = new QuitDialog(gameLoop.getScreenManager(), this);
        gameLoop.getGUIManager().addDialog(quitDialog);

        createGameActions();

    }

    /**
     * Gather game wide input.
     */
    public void gatherInput() {
        gatherSpecialInput();
  }

    /**
     * Gather special system input from the player (like pressing on
     * exit or pause game)
     */
    private void gatherSpecialInput() {

        if (pause.isPressed()) {
            setPaused(!paused);
        }

        if (exit.isPressed()) {
            exit.reset(); // Reset since the focus will pass to the dialog
            setPaused(true);
            gameLoop.getScreenManager().showCursor(true);
            quitDialog.setVisible(true);
            quitDialog.requestFocus();
        }
    }

    /**
     * Returns true if the user quitted the game.
     * @return  True if user quitted the game.
     */
    public boolean isQuit() {
        return quit;
    }

    /**
     * Set the quit state.
     * @param quit  True if quit false otherwise.
     */
    public void setQuit(boolean quit) {
        // Return the focus to the game frame
        gameLoop.getScreenManager().getFullScreenWindow().requestFocus();
        this.quit = quit;
        setPaused(false);
    }

    /**
     * Returns true if the game is paused.
     * @return  True if the game is paused.
     */
    public boolean isPaused() {
        return paused;
    }

    /**
     * Set the paused state of the game. In networked game the game
     * cannot be paused.
     * @param paused  True to pause, false to continue.
     */
    public void setPaused(boolean paused) {
        // Pause the game if not network game
        if (!gameLoop.isNetworkGame()) {
            this.paused = paused;
            gameLoop.getScreenManager().showCursor(paused);
            pause.reset();
        }
    }


    /* Implement KeyListener interface */

    /**
```

```
     * Find the GameAction for the key and if found call
     * its <code>press</code> method.
     */
    public void keyPressed(KeyEvent event) {
        int keyCode = event.getKeyCode();
        GameAction gameAction = (GameAction)
            keyCodeToGameAction.get(new Integer(keyCode));

        if (gameAction != null) {
            gameAction.press();
        }

        event.consume();

    }

    /**
     * Find the GameAction for the key and if found call
     * its <code>release</code> method.
     */
    public void keyReleased(KeyEvent event) {

        int keyCode = event.getKeyCode();
        GameAction gameAction = (GameAction)
            keyCodeToGameAction.get(new Integer(keyCode));

        if (gameAction != null) {
            gameAction.release();
        }

        event.consume();

    }

    /**
     * Ignore the key typed events
     */
    public void keyTyped(KeyEvent event) {
        event.consume();
    }

    /**
     * Creates and inserts to the map the game actions.
     */
    private void createGameActions() {

        exit = new GameAction();
        pause = new GameAction();
        moveLeft = new GameAction();
        moveRight = new GameAction();
        moveUp = new GameAction();
        moveDown = new GameAction();
        fireBullet = new GameAction();
        fireMissile = new GameAction();

        keyCodeToGameAction = new HashMap();

        keyCodeToGameAction.put(
                new Integer(KeyEvent.VK_ESCAPE), exit);
        keyCodeToGameAction.put(
                new Integer(KeyEvent.VK_P), pause);
        keyCodeToGameAction.put(
                new Integer(KeyEvent.VK_LEFT), moveLeft);
        keyCodeToGameAction.put(
                new Integer(KeyEvent.VK_RIGHT), moveRight);
        keyCodeToGameAction.put(
                new Integer(KeyEvent.VK_UP), moveUp);
        keyCodeToGameAction.put(
                new Integer(KeyEvent.VK_DOWN), moveDown);
        keyCodeToGameAction.put(
                new Integer(KeyEvent.VK_SPACE), fireBullet);
        keyCodeToGameAction.put(
                new Integer(KeyEvent.VK_CONTROL), fireMissile);

    }
}
```

```java
package game.input;

/**
 * The <code>GameAction</code> class represents a game
 * action key, like firing, moving left, etc.
 * It holds the state of the game action (e.g., pressed/released).
 */
public class GameAction {

    private final static int STATE_RELEASED = 0;
    private final static int STATE_PRESSED  = 1;

    private int state;  // The game action state

    /**
     * No parameters constructor.
     */
    public GameAction() {
        reset();
    }

    /**
     * Press the game action - mark as pressed.
     */
    public void press() {
        state = STATE_PRESSED;
    }

    /**
     * Release the game action - mark as released.
     */
    public void release() {
        state = STATE_RELEASED;
    }

    /**
     * Returns true if this <code>GameAction</code> is pressed.
     * @return True if this <code>GameAction</code> is pressed.
     */
    public boolean isPressed() {
        return (state == STATE_PRESSED);
    }

    /**
     * Resets this object state. Default is released.
     */
    public void reset() {
        state = STATE_RELEASED;
    }

}
```

```
package game.network;

/**
 * The <code>InvalidLoginException</code> is thrown by the server
 * side if the user tried to login with the wrong user name or password
 * and by the client if the ticket (the session id) received from server
 * was null.
 */
public class InvalidLoginException extends Exception {

  public InvalidLoginException() {
    super("Invalid login information");
  }

  public InvalidLoginException(String message) {
      super(message);
  }

}
```

```
package game.network.client;

/**
 * This interface hold the JNDI names of the ejbs
 * for accessing from standalone client.
 */
public interface ClientJNDINames {

    public final static String SIGN_IN_BEAN = "SignInBean";

  public final static String PLAYER_BEAN = "PlayerBean";

  public final static String ONLINE_PLAYER_BEAN = "OnlinePlayerBean";

  public final static String HIGH_SCORES_BEAN = "HighScoresBean";

  public final static String TOPIC_CONNECTION_FACTORY =
      "jms/TopicConnectionFactory";

  public final static String INVITATION_TOPIC =
      "jms/invitationTopic";

}
```

```java
package game.network.client;

import game.gamestate.GameState;
import game.network.packet.Packet;

/**
 * The <code>GameNetworkManager</code> interface defines the methods
 * that a network manager for the <b>running game</b> should implement.
 * The various game components access the network methods through this
 * interface only to make it easy to create different network managers
 * in the future.
 */
public interface GameNetworkManager {

    /**
     * Gather network input relevant to the game state.
     * The network manager should callback the game
     * state's <code>gatherInput</code> method.
     * @param gameState Current game state.
     */
    public void gatherInput(GameState gameState);

    /**
     * Send packet to the network player.
     * @param packet  Packet to send.
     */
    public void sendPacket(Packet packet);

    /**
     * Handle incoming packet. The implementing manager usually
     * queue the incoming packets to be processed by the
     * <code>GameState</code> in the gather input stage.
     * @param packet
     */
    public void handlePacket(Packet packet);

    /**
     * Returns the first packet in the incomig packets queue
     * and remove it from the queue.
     */
    public Packet getNextPacket();

    /**
     * Returns this player session id.
     */
    public Long getSenderId();

    /**
     * Returns the network player session id.
     */
    public Long getReceiverId();

    /**
     * Returns true if this user initiated the network game (i.e., sent
     * the invitation to play).
     */
    public boolean isInviter();

    /**
     * Do some cleanup before exiting.
     */
    public void cleanup();

}
```

```java
package game.network.client;

import java.util.*;

import javax.jms.JMSException;

import game.gamestate.GameState;
import game.network.packet.Packet;
import game.network.packet.PlayerQuitPacket;
import game.util.Logger;

/**
 * The <code>J2EEGameNetworkManager</code> implements the
 * <code>GameNetworkManager</code> interface and uses JMS
 * and J2EE API for the communication.
 */
public class J2EEGameNetworkManager implements GameNetworkManager {

    private JMSGameMessageHandler jmsGameMessageHandler;
    private PacketsSenderThread sender;

    private List inputQueue;    // List of incoming Packets
    private Collection outputQueue; // Collection of outgoing Packets

    private boolean inviter;
    private Long senderId;
    private Long receiverId;

    /**
     * Construct the J2EEGameNetworkManager
     * @param jmsGameHandler  JMS messages handler
     * @param senderId    User session id
     * @param receiverId  Network player session id
     * @param inviter   True if this machine is the inviter
     */
    public J2EEGameNetworkManager(
            JMSGameMessageHandler jmsGameHandler,
            Long senderId, Long receiverId, boolean inviter) {

        jmsGameHandler.setGameNetworkManager(this);
        this.jmsGameMessageHandler = jmsGameHandler;
        this.senderId = senderId;
        this.receiverId = receiverId;
        this.inviter = inviter;
        this.inputQueue = new ArrayList();
        this.outputQueue = new ArrayList();

        // Create and start the PacketsSenderThread
        sender = new PacketsSenderThread(outputQueue, jmsGameHandler);
        sender.start();

    }

    /**
     * Iterate on the incoming packets queue and pass each packet
     * to the game state to handle. If the game state consumed the
     * packet, remove it from the queue.
     * We remove the packet only if it's consumed since it might
     * happen that the packet is for the next state.
     */
    public void gatherInput(GameState gameState) {

        // Synchronize on the input queue
        synchronized (inputQueue) {
            Iterator inputQueueItr = inputQueue.iterator();
            while (inputQueueItr.hasNext()) {
                Packet packet = (Packet) inputQueueItr.next();

                gameState.handlePacket(packet);

                if (packet.isConsumed()) {
                    inputQueueItr.remove();
                }
            }
        }
    }

    /**
     * @return First packet in the input queue and removes the packet.
     * Null if no packet in the input queue.
     */
    public Packet getNextPacket() {
        Packet ret = null;
        synchronized (inputQueue) {
            if (!inputQueue.isEmpty()) {
                ret = (Packet)inputQueue.get(0);
                inputQueue.remove(0);
            }

            return ret;
        }
    }

    /**
     * Add the packet to the output queue and notify the thread
     * waiting on the output queue monitor.
     */
    public void sendPacket(Packet packet) {

    synchronized (outputQueue) {
        outputQueue.add(packet);

        outputQueue.notifyAll();
    }

    }

    /**
     * Add message to the input queue to be proccessed by the
     * game state when gather input is called.
     */
    public void handlePacket(Packet packet) {
        synchronized(inputQueue) {
            inputQueue.add(packet);
        }
    }

    /**
     * Returns the user session id.
     */
    public Long getSenderId() {
        return this.senderId;
    }

    /**
     * Returns the network user session id.
     */
    public Long getReceiverId() {
        return this.receiverId;
    }

    /**
     * Returns true if this machine initialized the network game.
     */
    public boolean isInviter() {
        return this.inviter;
    }

    /**
     * Clean and release resources. Send quit packet to the other
     * player if game is in progress.
     */
    public void cleanup() {

        // Stop the sender thread
        sender.stopSending();
        /** TODO: check if we need this check */
//        if (gameInProgress) {
        // Send player quit packet
System.out.println("\nSending playerQuitPacket\n");
        PlayerQuitPacket packet = new PlayerQuitPacket(senderId, receiverId);
        try {
            jmsGameMessageHandler.sendMessage(packet);
        }
        catch (JMSException jmse) {
            Logger.exception(jmse);
        }

//        }
    }
}
```

```java
package game.network.client;

import java.util.*;

import game.GameMenu;
import game.highscore.HighScore;
import game.network.InvalidLoginException;
import game.network.packet.*;
import game.network.server.ejb.*;
import game.util.Logger;

import javax.jms.*;
import javax.naming.*;
import javax.rmi.PortableRemoteObject;

/**
 * The <code>J2EENetworkManager</code> implements the
 * <code>NetworkManager</code> interface and uses JMS
 * and J2EE API for the communication.
 */
public class J2EENetworkManager implements NetworkManager {

    private GameMenu gameMenu;

    private Long sessionId;  // Hold this user session id
    private Long receiverId; // Session id of the network player
    private String userName;
    private boolean inviter;
    private boolean acceptInvitations;

    private Connection jmsConnection;
    private Session jmsSession;

    private JMSMessageHandler jmsInvitationManager;
    private JMSGameMessageHandler jmsGameMessageHandler;

    /**
     * Cinstruct the J2EE network manager.
     * @param game  Reference to the game menu.
     */
    public J2EENetworkManager(GameMenu game) {
        this.gameMenu = game;
    }

    /**
     * Sets the invitation acceptance of the current logged user.
     * The user accepts invitation when she logs in and reject invitation
     * when she start playing. We update the online player bean with the new
     * status.
     * @param accept Treu if the user is ready to accept invitations to play.
     */
    public void acceptInvitations(boolean accept) throws NetworkException {

        if (this.sessionId == null) {
            throw new NetworkException("User is not logged in");
        }

        try {

            this.acceptInvitations = accept;
            // Update the online player bean
            OnlinePlayerHome onlineHome = (OnlinePlayerHome)
        EJBHelper.getEJBHome(JNDINames.ONLINE_PLAYER_BEAN,
                OnlinePlayerHome.class);

        OnlinePlayer onlinePlayer = onlineHome.findByPrimaryKey(sessionId);

        onlinePlayer.setAcceptInvitations(accept);

        }
        catch (Exception e) {
            Logger.exception(e);
            throw new NetworkException("Error updating invitation status: " +
                    e.getMessage());
        }

    }

    /**
     * @see NetworkManager#sendPacket(Packet)
     */
    public void sendPacket(Packet packet) {
        // Not in use
    }

    /**
     * Login to the server. Find the sign in bean and try to log
     * in with the username and password.
     * @see NetworkManager#login(String, String)
     */
    public Long login(String userName, String password)
        throws NetworkException, InvalidLoginException {

    try {
      SignInHome signInHome = (SignInHome) EJBHelper.getEJBHome(
              ClientJNDINames.SIGN_IN_BEAN, SignInHome.class);

      SignIn signIn = signInHome.create();

      // Try to login with the supplied user and password
      this.sessionId = signIn.login(userName, password);

      // If login was successful set the current logged user,
      // init the jms connection and return the user session id
      initJMSConnection();
      this.userName = userName;
      return sessionId;

    }
    catch (InvalidLoginException ile) {
        throw ile;
    }
    catch (Exception e) {
        Logger.exception(e);
      throw new NetworkException(e.getMessage());
    }

    }

    /**
```

```
     * Logout from the game server and close the jms connection.
     * @see NetworkManager#logout()
     */
    public void logout() throws NetworkException {

    try {
      SignInHome signInHome = (SignInHome) EJBHelper.getEJBHome(
              ClientJNDINames.SIGN_IN_BEAN, SignInHome.class);

      SignIn signIn = signInHome.create();

        signIn.logout(sessionId);

        // Close the jms connection
        jmsConnection.close();

    }
    catch (Exception e) {
        Logger.exception(e);
        throw new NetworkException("Error while trying to logout: " +
                e.getMessage());
    }
    finally {
        this.sessionId = null;
        this.userName = null;
        this.jmsConnection = null;
    }

    }

    /**
     * Register with a new user. Find the sign in bean and signup.
     * @see NetworkManager#signup(String, String, String)
     */
    public void signup(String userName, String password, String email)
        throws NetworkException {

    try {

      SignInHome signInHome = (SignInHome) EJBHelper.getEJBHome(
              ClientJNDINames.SIGN_IN_BEAN, SignInHome.class);

      SignIn signIn = signInHome.create();

      signIn.addUser(userName, password, email);

    }
    catch (Exception e) {
      e.printStackTrace();
      throw new NetworkException(e.getMessage());
    }
    }

    /**
     * @see NetworkManager#getAvailablePlayers()
     */
    public List getAvailablePlayers() throws NetworkException{
        List playersModels = new ArrayList();
        try {

            OnlinePlayerHome onlinePlayerHome = (OnlinePlayerHome)
            EJBHelper.getEJBHome(JNDINames.ONLINE_PLAYER_BEAN,
                    OnlinePlayerHome.class);

            // Get collection of available online players
            Collection result = onlinePlayerHome.findByAcceptInvitations();

            // Get the online players models
            Iterator itr = result.iterator();
            while (itr.hasNext()) {
                OnlinePlayer onlinePlayer = (OnlinePlayer)
                  PortableRemoteObject.narrow(itr.next(),
                          OnlinePlayer.class);

                // Add the player if it's not the current player
                if (!onlinePlayer.getPrimaryKey().equals(this.sessionId)) {
                    playersModels.add(onlinePlayer.getOnlinePlayerModel());
                }

            }
            return playersModels;
        }
        catch (Exception e) {
            Logger.exception(e);
            throw new NetworkException("Network error while trying to " +
                "get available players list. Please try again later.");
        }
    }

    /**
     * Handle incoming invitation packets.
     * @see NetworkManager#handlePacket(Packet)
     */
    public void handlePacket(Packet packet) {

        if (packet instanceof JMSInvitationPacket) {

            handleInvitation((JMSInvitationPacket) packet);

        }

    }

    /**
     * Handle invitation packet.
     * @param invitation  Invitation packet with the details.
     */
    private void handleInvitation(JMSInvitationPacket invitation) {

        if (invitation.cancelled) {
            gameMenu.invitationCancelled();
        }

        else if (invitation.isReply) {
            // It is a reply to previously sent invitation sent by this user

            if (receiverId == null) {
                // In case the invitation was cancelled or arrived from
                // the wrong user ignore it
                return;
            }
            try {
              if (invitation.accepted) {

                  // Set the destination of the jms handler to be the private
                  // destination of the invitee
                  jmsGameMessageHandler.setDestination(
                          invitation.replyToDestination);

                  // Set the receiver id
                  this.receiverId = invitation.senderId;
                  // Mark the user as the inviter
                  this.inviter = true;

              }

              gameMenu.invitationAccepted(invitation.accepted,
invitation.userName);
            }
            catch (JMSException jmse) {
                Logger.exception(jmse);
                Logger.showErrorDialog(gameMenu, "Unable to proccess " +
                    "invitation reply: " + jmse.getMessage());
            }

        } else {  // Received invitation from another player

            if (acceptInvitations) {
                // Wait for the response from the user
                gameMenu.invitationArrived(invitation);
            }
        }

    }

    /**
     * @see NetworkManager#sendInvitationReply(InvitationPacket, boolean)
     */
    public void sendInvitationReply(InvitationPacket originalInvitation,
            boolean accepted) throws NetworkException {

        try {
          JMSInvitationPacket  invitation =
              (JMSInvitationPacket)originalInvitation;

          // Create a new JMSInvitationPacket with the reply to address
          // of the jms gameMenu listener queue
          JMSInvitationPacket invitationReply = new JMSInvitationPacket(
                  this.sessionId, invitation.senderId, this.userName,
                  jmsGameMessageHandler.getPrivateQueue());

          invitationReply.isReply = true;

          if (accepted) {
              // user accepted the invitation
              invitationReply.accepted = true;

              // Set the JMSGameListener's destination for the online gameMenu
              // to the private queue of the network player

jmsGameMessageHandler.setDestination(invitation.replyToDestination);

              this.receiverId = invitation.senderId;
```

```java
              // Mark this user as invitee
              this.inviter = false;

          } else {
              invitationReply.accepted = false;
          }

        jmsInvitationManager.sendInvitationReply(invitationReply);
        gameMenu.setStartMultiplayer(accepted);

        }
        catch (JMSException jmse) {
            Logger.exception(jmse);
            throw new NetworkException(jmse.getMessage());
        }
    }

    /**
     * Send invitation to play network gameMenu to the user with
     * session id equals to <code>destinationId</code>
     * @param destinationId Session id of the target user
     * @see NetworkManager#sendInvitation(Long)
     */
    public void sendInvitation(Long destinationId)
        throws NetworkException {

        try {
            this.receiverId = destinationId;

            JMSInvitationPacket packet = new JMSInvitationPacket(
                    sessionId, destinationId, userName,
                    jmsGameMessageHandler.getPrivateQueue());

            jmsInvitationManager.sendInvitation(packet);
        }
        catch (JMSException jmsException) {
            Logger.exception(jmsException);
            throw new NetworkException("Network error while trying to " +
                "send invitation to play");
        }

    }

    /**
     * Cancel the last invitation sent by this user. Inform the
     * invitee of the cancellation and set the destination id to null.
     * @see NetworkManager#cancelInvitation()
     */
    public void cancelInvitation() throws NetworkException{

        JMSInvitationPacket cancelInvitation = new JMSInvitationPacket(
                this.sessionId, receiverId, this.userName, null);
        cancelInvitation.cancelled = true;

        this.receiverId = null;

        jmsInvitationManager.sendPacket(cancelInvitation);

    }

    /**
```

```
     * Initialize the JMS connection. Find the connection factory
     * and create a connection to the invitation topic.
     */
    private void initJMSConnection()
      throws JMSException, NamingException {

        Context context = new InitialContext();

        // Find the connection factory
        ConnectionFactory connectionFactory =
            (ConnectionFactory) context.lookup(
                    ClientJNDINames.TOPIC_CONNECTION_FACTORY);

        // Creat the connection
        jmsConnection = connectionFactory.createConnection();

        // Creat the session
        jmsSession = jmsConnection.createSession(false,
Session.AUTO_ACKNOWLEDGE);

        // Create the JMS handlers for the game
        jmsInvitationManager = new JMSMessageHandler(this, sessionId,
jmsSession);
        jmsGameMessageHandler = new JMSGameMessageHandler(jmsSession);

        // Start receiving messages
        jmsConnection.start();

    }

    /**
     * Returns a new <code>J2EEGameNetworkManager</code> to manage
     * the network in the running game.
     */
    public GameNetworkManager getGameNetworkManager() {
        return new J2EEGameNetworkManager(
                jmsGameMessageHandler, getSenderId(),
                getReceiverId(), isInviter());
    }

    /**
     * @see NetworkManager#getSenderId()
     */
    public Long getSenderId() {
        return this.sessionId;
    }

    /**
     * @see NetworkManager#getReceiverId()
     */
    public Long getReceiverId() {
        return this.receiverId;
    }

    /**
     * @see NetworkManager#isInviter()
     */
    public boolean isInviter() {
        return this.inviter;
    }
    /**
     * Post the player score to the server using the high scores bean.
     * @see NetworkManager#postHighScore(HighScore)
     */
    public void postHighScore(HighScore score) throws NetworkException {
        try {
            HighScoresHome highScoresHome= (HighScoresHome)
        EJBHelper.getEJBHome(
              JNDINames.HIGH_SCORES_BEAN, HighScoresHome.class);

            HighScores highScores = highScoresHome.create();

            highScores.postHighScore(score);

        }
        catch (Exception e) {
            Logger.exception(e);
            throw new NetworkException(e.getMessage());
        }

    }

    /**
     * @see NetworkManager#getTopTenScores()
     */
    public HighScore[] getTopTenScores() throws NetworkException {

        return getHighScores(1, 10);

    }

    /**
     * @see NetworkManager#getHighScores(int, int)
     */
    public HighScore[] getHighScores(int fromRank, int toRank)
        throws NetworkException {
        try {
            HighScoresHome highScoresHome= (HighScoresHome)
          EJBHelper.getEJBHome(
                ClientJNDINames.HIGH_SCORES_BEAN, HighScoresHome.class);

            HighScores highScores = highScoresHome.create();

            return highScores.getHighScores(fromRank, toRank);

        }
        catch (Exception e) {
            Logger.exception(e);
            throw new NetworkException(e.getMessage());
        }

    }

} // end class J2EENetworkManager
```

```java
package game.network.client;

import game.network.packet.*;
import game.util.Logger;

import javax.jms.*;

/**
 * The <code>JMSGameMessageHandler</code> is used by the
 * <code>GameNetworkManager</code> to send and receive messages
 * from the JMS server.
 */
public class JMSGameMessageHandler implements MessageListener {

    Session session;    // JMS session
    Queue privateQueue;   // Temporary private queue of the player
    Destination destination;  // Network player destination queue
    MessageConsumer gameConsumer;
    MessageProducer gameProducer;
    GameNetworkManager gameNetworkManager;

    /**
     * Create the jms handler for the game.
     * @param session JMS session.
     */
    public JMSGameMessageHandler(Session session) throws JMSException {

        this.session = session;

        // Create temporary queue for the online game messages
        privateQueue = session.createTemporaryQueue();

        // Create consumer and set this object as the listener
        gameConsumer = session.createConsumer(privateQueue);
        gameConsumer.setMessageListener(this);

    }

    /**
     * New message received from the JMS queue. send to the network
     * manager to handle.
     */
    public void onMessage(Message message) {

        if (message instanceof ObjectMessage) {
          try {
              Packet packet = (Packet)((ObjectMessage)message).getObject();
              while (gameNetworkManager == null) {
                  // Loop until the network manager is ready
                  // It might not be ready at the beginning of the game
                  try {
                      Thread.sleep(50);
                  }
                  catch (InterruptedException ie) {
                      // Ignore
                  }
              }

              gameNetworkManager.handlePacket(packet);
          }
          catch (JMSException jmse) {
              Logger.exception(jmse);
          }
        }
    }

    /**
     * Send the packet to the network player's queue.
     * @param packet  Packet to send.
     */
    public void sendMessage(Packet packet) throws JMSException {
        // Create new object message
        ObjectMessage message = session.createObjectMessage();

        message.setObject(packet);

        gameProducer.send(message);

    }

    /**
     * Sets the JMS destination and create a message producer for
     * the destination.
     * @param destination New JMS destination (of the network player)
     */
    public void setDestination(Destination destination) throws JMSException {
        this.destination = destination;
        this.gameProducer = session.createProducer(destination);

    }

    /**
     * Returns this player private listening queue for incoming
     * game messages.
     */
    public Destination getPrivateQueue() {
        return this.privateQueue;
    }

    /**
     * Sets the game network manager.
     */
    public void setGameNetworkManager(GameNetworkManager manager) {
        this.gameNetworkManager = manager;
    }


}
```

```java
package game.network.client;

import game.network.packet.*;
import game.util.Logger;

import javax.jms.*;
import javax.naming.*;

/**
 * The <code>JMSMessageHandler</code> handles JMS communication
 * for the network manager.
 */
public class JMSMessageHandler implements MessageListener {

    // JMS variables
    private Session session;
    private MessageProducer messageProducer;
    private MessageConsumer messageConsumer;

    private NetworkManager networkManager;

    /**
     * Construct the jms message handler.
     * @param networkManager  The network manager.
     * @param sessionId   The player session id to be used as the selector
     * @param session   JMS session to the game topic
     */
    public JMSMessageHandler(NetworkManager networkManager,
            Long sessionId, Session session) throws NamingException, JMSException
{

        this.networkManager = networkManager;
        this.session = session;
        initJMSConnection(sessionId);

    }

    /**
     * Find the destination topic and create message consumer and
     * message producer.
     * @param sessionId Session id of the current logged user to be used
     * as message selector.
     * @throws JMSException   If the creation of the consumer/producer fails
     * @throws NamingException  If the destination topic is not found
     */
    private void initJMSConnection(Long sessionId)
      throws JMSException, NamingException {

        Context context = new InitialContext();

        Destination topic = (Topic) context.lookup(
                ClientJNDINames.INVITATION_TOPIC);

        messageProducer = session.createProducer(topic);

        // The selector accepts only messages destined to the current user
        String selector = "ReceiverID = '" + sessionId + "'";
        messageConsumer = session.createConsumer(topic, selector);
        messageConsumer.setMessageListener(this);
    }

    /**
     * New message received from the JMS topic designed for this
     * user. Send to the network manager to handle.
     */
    public void onMessage(Message message) {

        if (message instanceof ObjectMessage) {
          try {

              Packet packet = (Packet)((ObjectMessage)message).getObject();
//System.out.println("JMSMessageListener received: " + packet);
              if (packet instanceof JMSInvitationPacket) {
                  // Set the reply to destination before delivering to
                  // the network manager
                  JMSInvitationPacket invitation = (JMSInvitationPacket)packet;
                  invitation.replyToDestination = message.getJMSReplyTo();

              }

              networkManager.handlePacket(packet);

          }
          catch (JMSException jmsException) {
              Logger.exception(jmsException);
          }
        }
    }

    /**
     * Send the input packet to the destination topic. Adds the destination
     * user id as "ReceiverID" string property for the message selector.
     * @param packet  Packet to send.
     */
    public void sendPacket(Packet packet) throws NetworkException {
        try {

            Message message = session.createObjectMessage(packet);

            // Add the receiver id for the jms message selector
            message.setStringProperty("ReceiverID",
packet.receiverId.toString());

            messageProducer.send(message);

        }
        catch (JMSException jmsException) {
            Logger.exception(jmsException);
            throw new NetworkException("Error while trying to send packet to " +
                    packet.receiverId);
        }
    }

    /**
     * Send invitation to play.
     * @param packet  Invitation packet.
     */
    public void sendInvitation(JMSInvitationPacket packet) throws JMSException {

        Message message = session.createObjectMessage(packet);
        // Add the receiver id for the jms message selector
        message.setStringProperty("ReceiverID", packet.receiverId.toString());

        // Set the reply to destination
        message.setJMSReplyTo(packet.replyToDestination);

        messageProducer.send(message);
    }

    /**
     * Send reply to invitation.
     * @param packet  Packet with the details.
     */
    public void sendInvitationReply(JMSInvitationPacket packet)
      throws JMSException {

        Message message = session.createObjectMessage(packet);

        // Add the receiver id for the jms message selector
        message.setStringProperty("ReceiverID", packet.receiverId.toString());

        if (packet.accepted) {
            message.setJMSReplyTo(packet.replyToDestination);
        }

        messageProducer.send(message);

    }

}
```

```java
package game.network.client;

import java.io.IOException;

/**
 * General NetworkException to ease the network exception
 * handling on the client side.
 */
public class NetworkException extends IOException {

    public NetworkException(String message) {
        super(message);
    }

}
```

```java
package game.network.client;

import game.highscore.HighScore;
import game.network.InvalidLoginException;
import game.network.packet.InvitationPacket;
import game.network.packet.Packet;

import java.util.List;

/**
 * The <code>NetworkManager</code> interface defines the methods
 * that a network manager for the <b>game menu</b> (before starting
 * to play) needs to implement.
 * The various game components access the network methods through this
 * interface only to make it easy to create different network managers
 * in the future.
 * The methods that are called from the game components should
 * throw the checked <code>NetworkException</code>.
 */
public interface NetworkManager {

    /**
     * Send a packet to the network.
     * @param packet  Packet to send.
     */
    public void sendPacket(Packet packet);

    /**
     * Handle incoming packet.
     * @param packet  Incoming packet.
     */
    public void handlePacket(Packet packet);

    /**
     * Login to the game server.
     * @param user     Username.
     * @param password  Password.
     * @return       Session id of the player.
     * @throws InvalidLoginException  If details are wrong.
     */
    public Long login(String user, String password)
      throws NetworkException, InvalidLoginException;

    /**
     * Logout of the server.
     */
    public void logout() throws NetworkException;

    /**
     * Register to the game server.
     * @param user     Username
     * @param password  Password
     * @param email    Email (optional)
     */
    public void signup(String user, String password, String email)
      throws NetworkException;

    /**
     * Send invitation to play to an online player.
     * @param sessionId Session id of the invitee.
     */
    public void sendInvitation(Long sessionId) throws NetworkException;

    /**
     * Cancel the invitation to the last user.
     */
    public void cancelInvitation() throws NetworkException;

    /**
     * Sets the accept invitations flag of the user in the server.
     * @param accept  True to accept, false to deny.
     */
    public void acceptInvitations(boolean accept)
      throws NetworkException;

    /**
     * Send a reply to an invitation.
     * @param originalInvitation  The invitation packet
     * @param accepted  True if accepted, false otherwise.
     */
    public void sendInvitationReply(InvitationPacket originalInvitation,
            boolean accepted) throws NetworkException;

    /**
     * Returns a list of <code>OnlinePlayerModel</code> with the
     * online players details.
     * @return  List of available players.
     */
    public List getAvailablePlayers() throws NetworkException;

    /**
     * Returns the logged user session id.
     */
    public Long getSenderId();

    /**
     * Returns the destination user session id.
     */
    public Long getReceiverId();

    /**
     * Returns true if this user initiated the network game (i.e., sent
     * the invitation to play).
     */
    public boolean isInviter();

    /**
     * Post the player score to the server.
     * @param score
     * @throws NetworkException
     */
    public void postHighScore(HighScore score) throws NetworkException;

    /**
     * Returns the top ten scores from the server.
     */
    public HighScore[] getTopTenScores() throws NetworkException;

    /**
     * Returns the high scores from place <code>fromRank</code>
     * to <code>toRank</code> inclusive.
     * @param fromRank  Starting rank.
     * @param toRank  Ending rank
     */
    public HighScore[] getHighScores(int fromRank, int toRank)
      throws NetworkException;

    /**
     * Returns a network manager for the running game.
     */
    public GameNetworkManager getGameNetworkManager()
      throws NetworkException;
}
```

```java
package game.network.client;

import game.network.packet.Packet;
import game.util.Logger;

import java.util.Collection;
import java.util.Iterator;

import javax.jms.JMSException;

/**
 * The packet sender thread is used to send game packets
 * in a different thread than the game loop thread.
 */
public class PacketsSenderThread extends Thread {

    /** Collection of outgoing packets */
    private Collection outputQueue;

    private JMSGameMessageHandler jmsMessageHandler;
    private boolean stopped = false;

    /**
     * Construct the packets sender.
     * @param outputQueue Queue for the outgoing packets.
     * @param jmsMessageHandler JMS sender to send the packets.
     */
    public PacketsSenderThread(Collection outputQueue,
            JMSGameMessageHandler jmsMessageHandler) {

        this.outputQueue = outputQueue;
        this.jmsMessageHandler = jmsMessageHandler;
    }

    /**
     * Loop untill stopped and send packets from the queue
     * when they are available.
     */
    public void run() {
        while (!stopped) {
            try {
              synchronized (outputQueue) {
                  while (outputQueue.isEmpty()) {
                        outputQueue.wait();
                  }

                  // Get and send all messages
                  Iterator itr = outputQueue.iterator();
                  while (itr.hasNext()) {
                      Packet packet = (Packet) itr.next();
                      jmsMessageHandler.sendMessage(packet);
                      itr.remove();
                  }

                  outputQueue.notifyAll();
                  Thread.yield();
              }
            }
            catch (InterruptedException ie) {
                // Ignore and continue
            }
            catch (JMSException jmse) {
                Logger.exception(jmse);
            }
        }
    }

    /**
     * Stop the thread.
     */
    public void stopSending() {
        stopped = true;
    }

}
```

```java
package game.network.packet;

import game.ship.weapon.BulletModel;

/**
 * The bullet packet is sent when one of the ship fires
 * a new bullet. The packet contains the bullet details
 * in a <code>BulletModel</code> object.
 *
 * @see game.ship.weapon.BulletModel
 */
public class BulletPacket extends Packet {

    // Holds the bullt details including the bullet owner
    private BulletModel bulletModel ;

    /**
     * Constructs a new <code>BulletPacket</code>.
     * @param senderId    Session id of the sender
     * @param receiverId  Session id of the target user
     * @param firingShipId  Id of the firing ship
     * @param bulletModel Bullet details
     */
    public BulletPacket(Long senderId, Long receiverId,
            int firingShipId, BulletModel bulletModel) {

        super(senderId, receiverId, firingShipId);
        this.bulletModel = bulletModel;
    }

    public BulletModel getBulletModel() {
        return this.bulletModel;
    }

}
```

```
package game.network.packet;

/**
 * <code>InvitationPacket</code> is used to invite player for an
 * online game and to send replies for invitations to play.
 */
public class InvitationPacket extends Packet {

    public String userName;  // User name of the sender
    public boolean isReply;  // Is it a reply to invitation
    public boolean accepted; // Is the invitation accepted
    public boolean cancelled; // Is the invitation cancelled

    /**
     * Construct a new <code>InvitationPackat</code>
     * @param senderId    Session id of the inviter
     * @param receiverId  Session id of the invitee
     * @param userName    Username of the inviter
     */
    public InvitationPacket(Long senderId, Long receiverId, String userName) {
        super(senderId, receiverId);
        this.userName = userName;
        this.cancelled = false;
    }

    public String toString() {
        return super.toString() +
          " User name: " + userName +
          " isReply: " + isReply +
          " accepted: " + accepted +
          " cancelled: " + cancelled;
    }

}
```

```java
package game.network.packet;

import javax.jms.Destination;

/**
 * This class extends the <code>InvitationPacket</code>
 * class to add the JMS reply destination.
 */
public class JMSInvitationPacket extends InvitationPacket {

    // We use the replyToDestination to save the jms reply
    // to destination and pass it from the network manager
    // to the jms connection manager but not over the network
    transient public Destination replyToDestination;

    public JMSInvitationPacket(Long senderID,
            Long receiverID, String userName, Destination destination) {

        super(senderID, receiverID, userName);
        this.replyToDestination = destination;

    }

}
```

```java
package game.network.packet;

import java.util.Collection;

/**
 * The <code>NewLevelPacket</code> is created and sent by the
 * controller player (the player whose computer is controlling
 * the random and special events in a network game) before a new
 * level is started. This packet contains details needed for the new level.
 */
public class NewLevelPacket extends Packet {

    private Collection enemyShipsModels;

    /**
     * Constructs a new <code>NewLevelPacket</code>. The handler id
     * is not set since the levels manager is waiting for this packet
     * on the other side.
     * @param senderId    Session id of the sender
     * @param receiverId  Session id of the target user
     * @param enemyShipsModels  Collection of <code>ShipModel</code>
     * objects of the enemy ships for the new level.
     */
    public NewLevelPacket(Long senderId, Long receiverId,
            Collection enemyShipsModels) {

        super(senderId, receiverId);
        this.enemyShipsModels = enemyShipsModels;
    }

    /**
     * Returns the enemy ships models.
     * @return  Collection of enemy ship models.
     */
    public Collection getEnemyShipsModels() {
        return this.enemyShipsModels;
    }
}
```

```java
package game.network.packet;

import java.io.Serializable;

/**
 * The <code>Packet</code> class is used as base class to deliver
 * messages between two game clients.
 */
public abstract class Packet implements Serializable {

    public Long senderId;    // Session id of the sender
    public Long receiverId;    // Session id of the target

    // Id of the game object who should handle this packet
    public int handlerId;
    private boolean consumed = false; // Is this packet consumed

    /**
     * Construct new packet with target handler id equals to
     * default meaning: no special or not yet exists object.
     * @param senderId    Session id of the sender
     * @param receiverId  Session id of the target user
     */
    public Packet(Long senderId, Long receiverId) {
        this(senderId, receiverId, -1);
    }

    /**
     * Construct new packet.
     * @param senderId    Session id of the sender
     * @param receiverId  Session id of the target user
     * @param handlerId   Id of the object that should handle this packet
     */
    public Packet(Long senderId, Long receiverId, int handlerId) {
        this.senderId = senderId;
        this.receiverId = receiverId;
        this.handlerId = handlerId;
    }

    /**
     * Returns true if this packet was marked as consumed
     * by some of the game objects.
     */
    public boolean isConsumed() {
        return consumed;
    }

    /**
     * Mark the <i>consumed</i> state of this packet. Generally
     * an object that handled the packet can mark it as consumed
     * so other objects won't have to check it.
     * @param consumed  Consumed state.
     */
    public void setConsumed(boolean consumed) {
        this.consumed = consumed;
    }

    public String toString() {

        return "Class: " + getClass() + " SenderID: " + senderId +
            " ReceiverID: " + receiverId;
    }

}
```

```java
package game.network.packet;

import game.network.client.GameNetworkManager;

/**
 * Each object in the game that creates or handles packet
 * should implement this interface.
 */
public interface PacketHandler {

    /**
     * Create and send packet(s) via the network manager.
     * @param netManager  Network manager.
     */
    public void createPacket(GameNetworkManager netManager);

    /**
     * Handle the incoming packet.
     * @param packet  Incoming packet.
     */
    public void handlePacket(Packet packet);

    /**
     * Returns the object network handler id that handles
     * the packets for this object. It might be the same object
     * or a different object.
     * @return  Id of the packet handlet that should handle
     * incoming packets.
     */
    public int getHandlerId();

}
```

```java
package game.network.packet;

/**
 * The <code>PlayerQuitPacket</code> is sent when the
 * user quits at the middle of an online game.
 */
public class PlayerQuitPacket extends Packet {

    /**
     * Constructs a new <code>PlayerQuitPacket</code>.
     * @param senderId    Session id of the sender
     * @param receiverId  Session id of the target user
     */
    public PlayerQuitPacket(Long senderId, Long receiverId) {
        super(senderId, receiverId);
    }

}
```

```java
package game.network.packet;

/**
 * The <code>PowerUpPacket</code> is sent whenever an enemy
 * ship drops a power up bonus.
 */
public class PowerUpPacket extends Packet {

    public float x, y;  // Location of the powerup sprite
    public int powerUp; // How much armor the bonus adds to the ship

    /**
     * Constructs a new <code>BulletPacket</code>.
     * @param senderId    Session id of the sender
     * @param receiverId  Session id of the target user
     * @param handlerId   Id of the network handler
     */
    public PowerUpPacket(Long senderId, Long receiverId,
            int handlerId, float x, float y, int powerUp) {

        super(senderId, receiverId, handlerId);
        this.x = x;
        this.y = y;
        this.powerUp = powerUp;
    }
}
```

```java
package game.network.packet;

import game.ship.ShipState;

/**
 * The <code>ShipPacket</code> is sent by ship (enemy or friendly)
 * and encapsulates the ship current state in a <code>ShipState</code> object.
 *
 * @see game.ship.ShipState
 */
public class ShipPacket extends Packet {

    // Current state of the ship sending this packet
    private ShipState shipState;

    /**
     * Constructs a new <code>ShipPacket</code>.
     * @param senderId    Session id of the sender
     * @param receiverId  Session id of the target user
     * @param handlerId   Id of the ship generating this packet
     * @param shipState   The ship's current state
     */
    public ShipPacket(Long senderId, Long receiverId,
            int handlerId, ShipState shipState) {

        super(senderId, receiverId, handlerId);
        this.shipState = shipState;

    }

    /**
     * Return the ship state object.
     * @return  ShipState object.
     */
    public ShipState getShipState() {
        return this.shipState;
    }

}
```

```
package game.network.packet;

/**
 * The <code>SystemPacket</code> is used as a general packet to
 * deliver short info (represented by an int) between the players.
 */
public class SystemPacket extends Packet {

    // Available types of a SystemPacket
    public static final int TYPE_READY_TO_PLAY = 1;

    private int type; // Type of the system packet

    /**
     * Construct a new <code>SystemPacket</code>.
     * @param senderId    Session id of the sender
     * @param receiverId  Session id of the target user
     * @param type      Type of the system message
     */
    public SystemPacket(Long senderId, Long receiverId, int type) {
        super(senderId, receiverId);
        this.type = type;
    }

    /**
     * Returns the system packet type
     * @return  packet type
     */
    public int getType() {
        return this.type;
    }

    public String toString() {
        return super.toString() +
          " type: " + type;

    }

}
```

```java
package game.network.packet;

/**
 * The <code>WeaponUpgradePacket</code> is sent whenever an enemy
 * ship drops a weapon upgrade bonus.
 */
public class WeaponUpgradePacket extends Packet {

    public float x, y;  // Location of the sprite
    public int weaponType;

    /**
     * Constructs a new <code>BulletPacket</code>.
     * @param senderId    Session id of the sender
     * @param receiverId  Session id of the target user
     * @param handlerId   Id of the network handler
     * @param weaponType  Type of the weapon
     */
    public WeaponUpgradePacket(Long senderId, Long receiverId,
            int handlerId, float x, float y, int weaponType) {

        super(senderId, receiverId, handlerId);
        this.x = x;
        this.y = y;
        this.weaponType = weaponType;
    }
}
```

```java
package game.network.server;

import game.network.server.ejb.JNDINames;

import java.sql.Connection;
import java.sql.SQLException;

import javax.ejb.EJBException;
import javax.naming.InitialContext;
import javax.sql.DataSource;

/**
 * Helper class to get and release database connections.
 */
public class DBHelper {

    /**
     * Returns a connection to the game statabase.
     * @return Connection to the game statabase.
     */
    public static Connection getConnection() {
        try {
            InitialContext ic = new InitialContext();
            DataSource ds = (DataSource) ic.lookup(JNDINames.DBName);

            return ds.getConnection();

        } catch (Exception exception) {
            throw new EJBException("Unable to connect to database. " +
                exception.getMessage());
        }
    }

    /**
     * Closes a connection.
     * @param connection Connection to close.
     */
    public static void releaseConnection(Connection connection) {

        if (connection != null) {
            try {
                connection.close();

            } catch (SQLException sqlException) {
                throw new EJBException("Error in releaseConnection: " +
                        sqlException.getMessage());
            }
        }
    }

}
```

```java
package game.network.server.ejb;

import java.rmi.RemoteException;

import javax.ejb.*;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.rmi.PortableRemoteObject;

public class EJBHelper {

    /**
     * Returns the EJB home interface.
     * @param jndiName    JNDI name of the bean.
     * @param homeClass   Home class of the bean.
     * @return  EJB home interface.
     * @throws NamingException  If name couldn't be found.
     */
    public static EJBHome getEJBHome(String jndiName, Class homeClass)
      throws NamingException {

        InitialContext initialContext = new InitialContext();

        Object objref = initialContext.lookup(jndiName);

        return (EJBHome) PortableRemoteObject.narrow(objref, homeClass);
    }

    /**
     * Returns the next sequence id for the specified table.
     * The next sequence id is taken form the
     * <code>SequenceFactory</code> ejb.
     * If no sequence is available for the input table, create it.
     * @param tableName Table name
     * @return  Next sequence for the input table.
     */
    public static Long getNextSeqId(String tableName)
        throws RemoteException, NamingException, CreateException {

    SequenceFactoryHome sequenceFactoryHome = (SequenceFactoryHome)
      getEJBHome(JNDINames.SEQUENCE_FACTORY_BEAN,
              SequenceFactoryHome.class);

    // find the sequence factory for the input table.
    // If not found create it.
    SequenceFactory sequenceFactory = null;
    try {
      sequenceFactory =
          sequenceFactoryHome.findByPrimaryKey(tableName);
    }
    catch (FinderException fe) {
        sequenceFactory =
          sequenceFactoryHome.create(tableName);
    }

    return sequenceFactory.getNextID();
    }

}
```

```java
package game.network.server.ejb;

import game.highscore.HighScore;

import java.rmi.RemoteException;

import javax.ejb.EJBObject;

/**
 * The <code>HighScores</code> EJB manages the scores
 * posted by the game players.
 */
public interface HighScores extends EJBObject {

  /**
   * Adds the posted score to the high scores table
   * @param score New score to add
   */
    public void postHighScore(HighScore score) throws RemoteException;

    /**
     * @return Array of HighScore objects containing the top ten
     * high scores.
     */
  public HighScore[] getTopTenScores() throws RemoteException;

  /**
   * @return Array of HighScore objects containing the scores
   * ranked between fromRank to toRank.
   * @throws RemoteException if exception occures or fromRank
   * is less then 1 or fromRank > toRank.
   */
  public HighScore[] getHighScores(int fromRank, int toRank)
    throws RemoteException;

}
```

```
package game.network.server.ejb;

import game.highscore.HighScore;
import game.network.server.DBHelper;

import java.rmi.RemoteException;
import java.sql.*;

import javax.ejb.*;

/**
 * The <code>HighScores</code> EJB manages the scores
 * posted by the game players.
 */
public class HighScoresBean implements SessionBean {

    private SessionContext sessionContext;

  public HighScoresBean() {
      // SessionBean class must implement an empty constructor
  }

  /* SessionBean implementation */

    public void ejbActivate() throws EJBException, RemoteException {
        // Not in use in stateless session beans
    }

    public void ejbPassivate() throws EJBException, RemoteException {
        // Not in use in stateless session beans
    }

    public void ejbRemove() throws EJBException, RemoteException {

    }

  public void setSessionContext(SessionContext sessionContext) {

    this.sessionContext = sessionContext;
  }


  /* Home interface implementation */

  public void ejbCreate() throws CreateException {

  }


  /* Implement business methods */

  /**
   * Adds the posted score to the high scores table.
   * @see HighScores#postHighScore(HighScore)
   */
  public void postHighScore(HighScore score) throws EJBException {

      Connection connection = null;
      try {
      Long scoreId = EJBHelper.getNextSeqId("high_score");

      connection = DBHelper.getConnection();

      PreparedStatement ps = connection.prepareStatement(
                  "INSERT INTO high_score " +
                  "(score_id, player_name, score, level) " +
                  "values(?, ?, ?, ?)");

          ps.setLong(1, scoreId.longValue());
          ps.setString(2, score.getPlayerName());
          ps.setLong(3, score.getScore());
          ps.setInt(4, score.getLevel());

          ps.executeUpdate();

          ps.close();

      }
        catch (Exception e) {
            throw new EJBException(e);
        }
        finally {
            DBHelper.releaseConnection(connection);
        }

  }

  /**
   * @see HighScores#getTopTenScores()
   */
  public HighScore[] getTopTenScores() {
      return getHighScores(1, 10);
  }

  /**
   * @see HighScores#getHighScores(int, int)
   */
  public HighScore[] getHighScores(int fromRank, int toRank)
      throws EJBException {

      Connection connection = null;
      try {
          if (fromRank < 1) {
              throw new IllegalArgumentException("Input out of bounds");
          }

          if (fromRank > toRank) {
              throw new IllegalArgumentException("First argument must be " +
                  "smaller or equal to the second argument");
          }

          HighScore[] highScores = new HighScore[toRank-fromRank+1];

          connection = DBHelper.getConnection();

        // Create SQL query that selects all the rows in the
        // high scores table ordered by score and level descending
        PreparedStatement ps = connection.prepareStatement(
                "SELECT player_name, score, level " +
                "FROM high_score " +
                "ORDER BY score DESC, level DESC");

        ResultSet rs = ps.executeQuery();

        int curRank = 1;
        int count = 0;
        while (rs.next() && curRank <= toRank) {
            if (curRank >= fromRank) {
                String playerName = rs.getString(1);
                long score = rs.getLong(2);
                int level = rs.getInt(3);

                highScores[count] =
                    new HighScore(playerName, score, level);
                count++;
            }
        }

        rs.close();
        ps.close();

        return highScores;

      }
        catch (SQLException sqlException) {
            throw new EJBException(sqlException);
        }
        finally {
            DBHelper.releaseConnection(connection);
        }
  }

}
```

```java
package game.network.server.ejb;

import java.rmi.RemoteException;

import javax.ejb.CreateException;
import javax.ejb.EJBHome;

/**
 * The <code>HighScores</code> EJB manages the scores
 * posted by the game players.
 */
public interface HighScoresHome extends EJBHome {

    /**
     * Create a new high score.
     */
    public HighScores create() throws RemoteException, CreateException;

}
```

```java
package game.network.server.ejb;

/**
 * This interface holds static variables with the JNDI
 * names of the EJBs and the database.
 */
public interface JNDINames {

  public final static String DBName =
    "java:comp/env/jdbc/gameDB";

  public final static String PLAYER_BEAN =
      "java:comp/env/ejb/Player";

  public final static String ONLINE_PLAYER_BEAN =
      "java:comp/env/ejb/OnlinePlayer";

  public final static String HIGH_SCORES_BEAN =
      "java:comp/env/ejb/HighScores";

  public final static String SEQUENCE_FACTORY_BEAN =
      "java:comp/env/ejb/SequenceFactory";

}
```

```java
package game.network.server.ejb;

import java.rmi.RemoteException;

import javax.ejb.EJBObject;

/**
 * The <code>OnlinePlayer</code> represents an online player
 * that is logged in the system.
 * Each online player has its own primary pk (the session id).
 */
public interface OnlinePlayer extends EJBObject {

    /**
     * Sets the desire of the online player to allow
     * invitations to be sent to him.
     * The players will only see the online players with the
     * accept flag set to true in the online players list.
     * @param accept  True to accept false not to.
     */
    public void setAcceptInvitations(boolean accept)
      throws RemoteException;

    /**
     * Returns a simple model of the online player which contains
     * its session id, name etc.
     * @return  OnlinePlayerModel of the player.
     * @see OnlinePlayerModel
     */
    public OnlinePlayerModel getOnlinePlayerModel()
      throws RemoteException;

}
```

```java
package game.network.server.ejb;

import game.network.server.DBHelper;

import java.rmi.RemoteException;
import java.sql.*;
import java.util.ArrayList;
import java.util.Collection;

import javax.ejb.*;

public class OnlinePlayerBean implements EntityBean {

    private EntityContext entityContext;

    private Long sessionId;   // Primary key
    private String userName;  // Foreign key
    private long sessionStartTime;
    private boolean acceptInvitations;

    public OnlinePlayerBean() {
        // Must implement no arguments constructor
    }

    /**
     * Creates a new online player.
     * @param userName  User name of the player.
     * @return  Primary key (session id).
     * @see OnlinePlayerHome#create(String)
     */
    public Long ejbCreate(String userName) throws CreateException {

        this.userName = userName;
        this.acceptInvitations = false;
        this.sessionStartTime = System.currentTimeMillis();

        Connection connection = null;
        try {
      sessionId = EJBHelper.getNextSeqId("online_player");

      connection = DBHelper.getConnection();

          PreparedStatement ps = connection.prepareStatement(
              "INSERT INTO online_player " +
            "(session_id, user_name, session_start_time, accept_invitations) " +
            "values(?, ?, ?, ?)");

          ps.setLong(1, sessionId.longValue());
          ps.setString(2, userName);
          ps.setLong(3, sessionStartTime);
          ps.setBoolean(4, acceptInvitations);

          ps.executeUpdate();

          ps.close();

          return sessionId;

        }
        catch (Exception exception) {
            throw new CreateException(exception.getMessage());
        }
        finally {
            DBHelper.releaseConnection(connection);
        }

    }

    // for the ejbCreate(String)
    public void ejbPostCreate(String userName) {
        // nothing to do
    }

    public void ejbActivate() throws EJBException, RemoteException {
        this.sessionId = (Long) entityContext.getPrimaryKey();
    }

    public void ejbPassivate() throws EJBException, RemoteException {
        sessionId = null;
    }

    /**
     * Load the online player's details from the database.
     */
    public void ejbLoad() throws EJBException, RemoteException {

        Connection connection = null;
        try {

            Long sessionId = (Long) entityContext.getPrimaryKey();

      connection = DBHelper.getConnection();

            PreparedStatement ps = connection.prepareStatement(
                    "SELECT session_id, user_name, " +
                    "session_start_time, accept_invitations " +
                    "FROM online_player " +
                    "WHERE session_id = ?");

            ps.setLong(1, sessionId.longValue());

            ResultSet rs = ps.executeQuery();

      if (rs.next()) {
          this.sessionId = new Long(rs.getLong(1));
          this.userName = rs.getString(2);
          this.sessionStartTime = rs.getLong(3);
          this.acceptInvitations = rs.getBoolean(4);
      }
      else {
          throw new EJBException("Session not found: " + sessionId);
      }

      rs.close();
      ps.close();

        }
        catch (SQLException sqlException) {
            throw new EJBException(sqlException);
        }
        finally {
            DBHelper.releaseConnection(connection);
        }

    }

    /**
     * Removes the online player from the database.
     * Should be called when the player logs out or quits the game.
     */
    public void ejbRemove() throws
      RemoveException, EJBException, RemoteException {

        Connection connection = null;
        try {
            Long sessionId = (Long) entityContext.getPrimaryKey();

      connection = DBHelper.getConnection();

            PreparedStatement ps = connection.prepareStatement(
                    "DELETE FROM online_player " +
                    "WHERE session_id = ?");

            ps.setLong(1, sessionId.longValue());

            ps.executeUpdate();

        }
        catch (SQLException sqlException) {
      throw new RemoveException("Error while trying to remove " +
          "online player id " + sessionId + "\n" +
          sqlException.getMessage());
        }
        finally {
            DBHelper.releaseConnection(connection);
        }

    }

    /**
     * Store the details to the database.
     */
    public void ejbStore() throws EJBException, RemoteException {

        Connection connection = null;
        try {
      connection = DBHelper.getConnection();

            PreparedStatement ps = connection.prepareStatement(
                    "UPDATE online_player " +
                    "SET user_name = ?, session_start_time = ?, " +
                    "accept_invitations = ? " +
                    "WHERE session_id = ?");

            ps.setString(1, userName);
            ps.setLong(2, sessionStartTime);
            ps.setBoolean(3, acceptInvitations);
            ps.setLong(4, sessionId.longValue());

            ps.executeUpdate();

            ps.close();
        }
        catch (SQLException sqlException) {
            throw new EJBException(sqlException);
        }
        finally {
            DBHelper.releaseConnection(connection);
        }

    }

    public void setEntityContext(EntityContext entityContext)
      throws EJBException, RemoteException {

    this.entityContext = entityContext;

    }

  public void unsetEntityContext() throws EJBException, RemoteException {

    entityContext = null;

  }

  /**
   * Find online player by primary key.
   */
    public Long ejbFindByPrimaryKey(Long sessionId)
      throws FinderException {

    boolean found = false;

    Connection connection = null;
    try {
      connection = DBHelper.getConnection();

      PreparedStatement ps = connection.prepareStatement(
          "SELECT session_id " +
          "FROM online_player " +
          "WHERE session_id = ?");

      ps.setLong(1, sessionId.longValue());

      ResultSet rs = ps.executeQuery();

      found = rs.next();

      rs.close();
      ps.close();

      if (found) {
          return sessionId;
      }
      else {
          throw new FinderException("Session " + sessionId + " doesn't exist");
      }

    }
    catch (SQLException sqlException) {
      throw new EJBException(sqlException);
    }
```

```
        finally {
            DBHelper.releaseConnection(connection);
        }
    }

    /**
     * @see OnlinePlayerHome#findByAcceptInvitations()
     */
    public Collection ejbFindByAcceptInvitations()
      throws FinderException {

    Collection result = new ArrayList();

    Connection connection = null;
    try {
      connection = DBHelper.getConnection();

      PreparedStatement ps = connection.prepareStatement(
          "SELECT session_id " +
          "FROM online_player " +
          "WHERE accept_invitations = ?");

      ps.setBoolean(1, true);

      ResultSet rs = ps.executeQuery();

      while(rs.next()) {
          Long sessionId = new Long(rs.getLong(1));
          result.add(sessionId);
      }

      rs.close();
      ps.close();

      return result;

    }
    catch (SQLException sqlException) {
      throw new EJBException(sqlException);
    }
        finally {
            DBHelper.releaseConnection(connection);
        }
  }

    public OnlinePlayerModel getOnlinePlayerModel() {
        return new OnlinePlayerModel(sessionId, userName, sessionStartTime);
    }


    /* Implement business methods */

    /**
     * @see OnlinePlayer#setAcceptInvitations(boolean)
     */
    public void setAcceptInvitations(boolean acceptInvitations) {
        this.acceptInvitations = acceptInvitations;
    }

}
```

```java
package game.network.server.ejb;

import java.rmi.RemoteException;
import java.util.Collection;

import javax.ejb.*;

public interface OnlinePlayerHome extends EJBHome {

    /**
     * Find online player by primary key (session id).
     */
    public OnlinePlayer findByPrimaryKey(Long pk)
    throws RemoteException, FinderException;

    /**
     * Find all the online players that accepts invitations.
     */
    public Collection findByAcceptInvitations()
      throws RemoteException, FinderException;

    /**
     * Create a new online player. Use the next sequence as
     * the session id.
     * @param userName  User name of the online player.
     */
    public OnlinePlayer create(String userName)
    throws RemoteException, CreateException;

}
```

```java
package game.network.server.ejb;

import java.io.Serializable;

/**
 * The <code>OnlinePlayerModel</code> holds data about
 * a single online player to be sent over the network.
 */
public class OnlinePlayerModel implements Serializable {

    private Long sessionId;
    private String userName;
    private long sessionStartTime;

    /**
     * Construct the object.
     * @param sessionId   Session id of the player.
     * @param userName    User name of the player.
     * @param sessionStartTime  Start time of the session.
     */
    public OnlinePlayerModel(Long sessionId, String userName,
            long sessionStartTime) {

        this.sessionId = sessionId;
        this.userName = userName;
        this.sessionStartTime = sessionStartTime;

    }

    /**
     * @return Returns the sessionId.
     */
    public Long getSessionId() {
        return sessionId;
    }
    /**
     * @param sessionId The sessionId to set.
     */
    public void setSessionId(Long sessionId) {
        this.sessionId = sessionId;
    }
    /**
     * @return Returns the sessionStartTime.
     */
    public long getSessionStartTime() {
        return sessionStartTime;
    }
    /**
     * @param sessionStartTime The sessionStartTime to set.
     */
    public void setSessionStartTime(long sessionStartTime) {
        this.sessionStartTime = sessionStartTime;
    }
    /**
     * @return Returns the userName.
     */
    public String getUserName() {
        return userName;
    }
    /**
     * @param userName The userName to set.
     */
    public void setUserName(String userName) {
        this.userName = userName;
    }
}
```

```java
package game.network.server.ejb;

import java.rmi.RemoteException;

import javax.ejb.EJBObject;

/**
 * The <code>Player</code> EJB holds the data of a registered player.
 */
public interface Player extends EJBObject {

    /**
     * Returns the user name (the primary key).
     * @return  The user name.
     */
    public String getUserName() throws RemoteException;

    /**
     * Returns the user's passsword.
     * @return The user's passsword.
     */
    public String getPassword() throws RemoteException;

    /**
     * Sets the user's password.
     * @param password  Password to set.
     */
    public void setPassword(String password) throws RemoteException;

    /**
     * Returns the user's email.
     * @return The user's email.
     */
    public String getEmail() throws RemoteException;

    /**
     * Sets the user's email.
     * @param email Email to set.
     */
    public void setEmail(String email) throws RemoteException;


}
```

```java
package game.network.server.ejb;

import game.network.server.DBHelper;

import java.rmi.RemoteException;
import java.sql.*;

import javax.ejb.*;

/**
 * The <code>Player</code> EJB holds the data of a registered player.
 */
public class PlayerBean implements EntityBean {

    private EntityContext entityContext;
//    private Connection connection;

    private String userName;
    private String password;
    private String email;

    public PlayerBean() {
        // Must implement empty constructor
    }

    /**
     * @see PlayerHome#create(String, String)
     */
    public String ejbCreate(String userName, String password)
        throws CreateException {

        return ejbCreate(userName, password, null);

    }

    /**
     * For the ejbCreate(String, String)
     */
    public void ejbPostCreate(String userName, String password)
    throws CreateException {
        // do nothing
    }

    /**
     * @see PlayerHome#create(String, String, String)
     */
    public String ejbCreate(String userName, String password, String email)
      throws CreateException {

        this.userName = userName;
        this.password = password;
        this.email = email;

        Connection connection = null;
        try {
            connection = DBHelper.getConnection();
      PreparedStatement ps = connection.prepareStatement(
          "INSERT INTO player " +
          "(user_name, password, email) " +
          "VALUES(?, ?, ?)");
      ps.setString(1, userName);
      ps.setString(2, password);
      ps.setString(3, email);

      ps.executeUpdate();

      ps.close();

          return userName;

    }
    catch (SQLException sqlException) {
      throw new CreateException(sqlException.getMessage());
    }
        finally {
            DBHelper.releaseConnection(connection);
        }

    }

    /**
     * For the ejbCreate(String, String, String)
     */
    public void ejbPostCreate(String userName, String password,
            String email) throws CreateException {
        // do nothing
    }

    /**
     * Set the primary key.
     */
    public void ejbActivate() throws EJBException, RemoteException {
        userName = (String) entityContext.getPrimaryKey();
    }

    /**
     * Unset the primary key.
     */
    public void ejbPassivate() throws EJBException, RemoteException {
        userName = null;
    }

    /**
     * Load the details from the database.
     */
    public void ejbLoad() throws EJBException, RemoteException {

        Connection connection = null;
        try {
            // Get the primary key
            String userName = (String) entityContext.getPrimaryKey();

            connection = DBHelper.getConnection();
      PreparedStatement ps = connection.prepareStatement(
          "SELECT user_name, password, email " +
          "FROM player " +
          "WHERE user_name = ?");

      ps.setString(1, userName);

      ResultSet rs = ps.executeQuery();
      if (rs.next()) {
          this.userName = rs.getString(1);
          this.password = rs.getString(2);
          this.email = rs.getString(3);
      }
      else {
          throw new EJBException("No such player: " + userName);
      }

      rs.close();
      ps.close();

    }
    catch (SQLException sqlException) {
      throw new EJBException(sqlException);
    }
        finally {
            DBHelper.releaseConnection(connection);
        }

    }

    /**
     * Remove the player from the database.
     */
    public void ejbRemove() throws RemoveException, EJBException,
            RemoteException {

        Connection connection = null;
        try {
            // Get the primary key
            String userName = (String) entityContext.getPrimaryKey();

            connection = DBHelper.getConnection();
      PreparedStatement ps = connection.prepareStatement(
          "DELETE FROM player " +
          "WHERE user_name = ?");

      ps.setString(1, userName);

      ps.executeUpdate();

      ps.close();

    }
    catch (SQLException sqlException) {
      throw new RemoveException("Error while trying to remove player " +
              userName + "\n" + sqlException.getMessage());
    }
        finally {
            DBHelper.releaseConnection(connection);
        }

    }

    /**
     * Store the details to the database.
     */
    public void ejbStore() throws EJBException, RemoteException {
        Connection connection = null;
        try {
            // Get the primary key
            String userName = (String) entityContext.getPrimaryKey();

            connection = DBHelper.getConnection();
      PreparedStatement ps = connection.prepareStatement(
          "UPDATE player " +
          "SET password = ?, email = ? " +
          "WHERE user_name = ?");

      ps.setString(1, password);
      ps.setString(2, email);
      ps.setString(3, userName);

      ps.executeUpdate();

      ps.close();

    }
    catch (SQLException sqlException) {
      throw new EJBException(sqlException);
    }
        finally {
            DBHelper.releaseConnection(connection);
        }

    }

    public void setEntityContext(EntityContext entityContext)
        throws EJBException, RemoteException {

        this.entityContext = entityContext;
    }

    public void unsetEntityContext() throws EJBException, RemoteException {

        entityContext = null;

    }

    /**
     * Find by primary key (user name).
     */
    public String ejbFindByPrimaryKey(String userName)
        throws FinderException {

        boolean found = false;

        Connection connection = null;
        try {
            connection = DBHelper.getConnection();
      PreparedStatement ps = connection.prepareStatement(
          "SELECT user_name " +
          "FROM player " +
          "WHERE user_name = ?");

      ps.setString(1, userName);

      ResultSet rs = ps.executeQuery();
      found = rs.next();

      rs.close();
      ps.close();

      if (found) {
          return userName;
      }
      else {
          throw new FinderException("User " + userName + " doesn't exist");
      }

    }
    catch (SQLException sqlException) {
      throw new EJBException(sqlException);
    }
        finally {
            DBHelper.releaseConnection(connection);
        }
    }

    /* Implement business methods */

    public String getUserName() {
        return userName;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

}
```

```java
package game.network.server.ejb;

import java.rmi.RemoteException;

import javax.ejb.*;

public interface PlayerHome extends EJBHome {

    /**
     * Find player by primary key (user name)
     */
    public Player findByPrimaryKey(String pk)
      throws RemoteException, FinderException;

    /**
     * Create a new user.
     * @param pk       User name
     * @param password  Password
     */
    public Player create(String pk, String password)
      throws RemoteException, CreateException;

    /**
     * Create a new user.
     * @param pk       User name
     * @param password  Password
     * @param email    Email
     */
    public Player create(String pk, String password, String email)
      throws RemoteException, CreateException;

}
```

```java
package game.network.server.ejb;

import java.rmi.RemoteException;

import javax.ejb.EJBObject;

/**
 * The <code>SequenceFactory</code> is used to generate unique
 * sequence id for each table name (or any name).
 */
public interface SequenceFactory extends EJBObject {

    /**
     * Returns the next dequence id.
     * @return  The ext sequence id.
     */
    public Long getNextID() throws RemoteException;

}
```

```java
package game.network.server.ejb;

import game.network.server.DBHelper;

import java.rmi.RemoteException;
import java.sql.*;

import javax.ejb.*;

/**
 * The <code>SequenceFactory</code> is used to generate unique
 * sequence id for each table name (or any name).
 */
public class SequenceFactoryBean implements EntityBean {

    private EntityContext entityContext;

    private String tableName; // Primary key
    private Long nextID;

    /**
     * Create new sequence generator.
     * @param tableName Primary key.
     */
    public String ejbCreate(String tableName) throws CreateException {
        this.tableName = tableName;
        this.nextID = new Long(1);  // Start from id 1

        Connection connection = null;
        try {
            connection = DBHelper.getConnection();
            PreparedStatement ps = connection.prepareStatement(
                    "INSERT INTO sequence_factory " +
                    "(table_name, next_id) " +
                    "VALUES(?, ?)");

            ps.setString(1, tableName);
            ps.setLong(2, nextID.longValue());

            ps.executeUpdate();

            ps.close();

            return tableName;

        }
        catch (SQLException sqlException){
            throw new CreateException(sqlException.getMessage());
        }
        finally {
            DBHelper.releaseConnection(connection);
        }

    }

    /**
     * For the ejbCreate(String)
     */
    public void ejbPostCreate(String tableName) {
    }

    /**
     * Set the primary key.
     */
    public void ejbActivate() throws EJBException, RemoteException {
        this.tableName = (String) entityContext.getPrimaryKey();
    }

    /**
     * Unset the primary key.
     */
    public void ejbPassivate() throws EJBException, RemoteException {
        this.tableName = null;
    }

    /**
     * Load the details from the database.
     */
    public void ejbLoad() throws EJBException, RemoteException {
        Connection connection = null;
        try {
            String tableName = (String) entityContext.getPrimaryKey();

            connection = DBHelper.getConnection();
            PreparedStatement ps = connection.prepareStatement(
                    "SELECT table_name, next_id " +
                    "FROM sequence_factory " +
                    "WHERE table_name = ?");

            ps.setString(1, tableName);

            ResultSet rs = ps.executeQuery();

      if (rs.next()) {
          this.tableName = rs.getString(1);
          this.nextID = new Long(rs.getLong(2));
      }
      else {
          throw new EJBException("Table name not found: " + tableName);
      }

      rs.close();
      ps.close();

        }
        catch (SQLException sqlException) {
            throw new EJBException(sqlException);
        }
        finally {
            DBHelper.releaseConnection(connection);
        }

    }

    /**
     * Remove from the database.
     */
    public void ejbRemove() throws RemoveException, EJBException,
            RemoteException {
```

```
        Connection connection = null;
        try {
            String tableName = (String) entityContext.getPrimaryKey();

            connection = DBHelper.getConnection();
            PreparedStatement ps = connection.prepareStatement(
                    "DELETE FROM sequence_factory " +
                    "WHERE table_name = ?");

            ps.setString(1, tableName);

            ps.executeUpdate();

        }
        catch (SQLException sqlException) {
      throw new RemoveException("Error while trying to remove " +
          tableName + " from sequence_factory.\n" +
          sqlException.getMessage());
        }
        finally {
            DBHelper.releaseConnection(connection);
        }

    }

    /**
     * Store details to the database.
     */
    public void ejbStore() throws EJBException, RemoteException {
        Connection connection = null;
        try {
            connection = DBHelper.getConnection();
            PreparedStatement ps = connection.prepareStatement(
                    "UPDATE sequence_factory " +
                    "SET next_id = ? " +
                    "WHERE table_name = ?");

            ps.setLong(1, nextID.longValue());
            ps.setString(2, tableName);

            ps.executeUpdate();

            ps.close();

        }
        catch (SQLException sqlException) {
            throw new EJBException(sqlException);
        }
        finally {
            DBHelper.releaseConnection(connection);
        }

    }

    public void setEntityContext(EntityContext entityContext)
        throws EJBException, RemoteException {
        this.entityContext = entityContext;
    }

    public void unsetEntityContext() throws EJBException, RemoteException {
        this.entityContext = null;
```

```java
    }

    /**
     * Find the bean by the primary key.
     * @param tableName Primary key
     */
    public String ejbFindByPrimaryKey(String tableName) throws FinderException {
        boolean found = false;

        Connection connection = null;
        try {
            connection = DBHelper.getConnection();
      PreparedStatement ps = connection.prepareStatement(
          "SELECT table_name " +
          "FROM sequence_factory " +
          "WHERE table_name = ?");

      ps.setString(1, tableName);

      ResultSet rs = ps.executeQuery();

      found = rs.next();

      rs.close();
      ps.close();

      if (found) {
          return tableName;
      }
      else {
          throw new FinderException("Table " + tableName + " doesn't exist");
      }

    }
    catch (SQLException sqlException) {
      throw new EJBException(sqlException);
    }
        finally {
            DBHelper.releaseConnection(connection);
        }
    }


    /* Implement business methods */

    /**
     * @see SequenceFactory#getNextID()
     */
    public Long getNextID() {

        Long id = new Long(nextID.longValue());

        // Increment the id by 1
        nextID = new Long(nextID.longValue() + 1);

        return id;

    }

}
```

```java
package game.network.server.ejb;

import java.rmi.RemoteException;

import javax.ejb.*;

/**
 * The <code>SequenceFactory</code> is used to generate unique
 * sequence id for each table name (or any name).
 */
public interface SequenceFactoryHome extends EJBHome {

    /**
     * Create a new sequence bean.
     * @param pk  Usually a table name but can be any unique string.
     */
    public SequenceFactory create(String pk)
      throws RemoteException, CreateException;

    /**
     * Find bean by primary key.
     */
    public SequenceFactory findByPrimaryKey(String pk)
        throws RemoteException, FinderException;

}
```

```java
package game.network.server.ejb;

import game.network.InvalidLoginException;

import java.rmi.RemoteException;

import javax.ejb.EJBObject;

/**
 * The signin bean is used for users validation, logout and signups.
 */
public interface SignIn extends EJBObject {

    /**
     * Try to log in with the supplied username and password.
     * @param userName  User name
     * @param password  Passwrod
     * @return  Session id if logged in successfully
     * @throws InvalidLoginException If the user or password are wrong.
     */
  public Long login(String userName, String password)
    throws RemoteException, InvalidLoginException;

  /**
   * Logout of the system.
   * @param sessiondId  Session id to finish.
   */
  public void logout(Long sessiondId)
    throws RemoteException;

  /**
   * Signup a new user.
   * @param userName  User name
   * @param password  Password
   * @param email   Email of the user (may be empty).
   * @throws RemoteException
   */
  public void addUser(String userName, String password, String email)
    throws RemoteException;

}
```

```java
package game.network.server.ejb;

import game.network.InvalidLoginException;

import javax.ejb.*;

/**
 * The signin bean is used for users validation, logout and signups.
 */
public class SignInBean implements SessionBean {

  private SessionContext sessionContext;

  /* SessionBean class must implement an empty constructor */
  public SignInBean() {}

  /* SessionBean interface implementation */

  public void ejbActivate() {}
  public void ejbPassivate() {}

  public void ejbRemove() {}

  public void setSessionContext(SessionContext sessionContext) {

    this.sessionContext = sessionContext;
  }


  /* SignInHome interface implementation */

  public void ejbCreate() throws CreateException {

  }

  /* SignIn interface implementation (business methods) */

  /**
   * Login with the supplied username and password.
   */
  public Long login(String userName, String password)
      throws EJBException, InvalidLoginException {

      try {

      PlayerHome playerHome = (PlayerHome)
        EJBHelper.getEJBHome(JNDINames.PLAYER_BEAN, PlayerHome.class);

      // create the enterprise bean instance
      Player player = playerHome.findByPrimaryKey(userName);

      String playerPassword = player.getPassword();
      if (!playerPassword.equals(password)) {
          throw new InvalidLoginException();
      }

      // Validation successful, create session
      OnlinePlayerHome onlineHome = (OnlinePlayerHome)
        EJBHelper.getEJBHome(JNDINames.ONLINE_PLAYER_BEAN,
                OnlinePlayerHome.class);
      OnlinePlayer onlinePlayer = onlineHome.create(userName);

      return (Long) onlinePlayer.getPrimaryKey();

      }
      catch (FinderException fe) {
          throw new InvalidLoginException();
      }
      catch (Exception ne) {
          throw new EJBException(ne.getMessage());
      }
  }

  /**
   * Logout - remove the session from the online players table
   * @param sessionId Session id to remove
   */
  public void logout(Long sessionId) {

      try {

      OnlinePlayerHome onlinePlayerHome = (OnlinePlayerHome)
        EJBHelper.getEJBHome(JNDINames.ONLINE_PLAYER_BEAN,
              OnlinePlayerHome.class);

          OnlinePlayer onlinePlayer =
              onlinePlayerHome.findByPrimaryKey(sessionId);

          onlinePlayer.remove();

      }
      catch (Exception exception) {
          throw new EJBException(exception.getMessage());
      }

  }

  /**
   * Create a new user accout
   * @param userName  Username
   * @param password  Password
   * @param email     Email
   */
  public void addUser(String userName, String password,
      String email) throws EJBException {

      try {

      PlayerHome playerHome = (PlayerHome)
        EJBHelper.getEJBHome(JNDINames.PLAYER_BEAN, PlayerHome.class);

      playerHome.create(userName, password, email);

    }
      catch (CreateException ce) {
          throw new EJBException("Can't create user " + userName +".\n"
                  + ce.getMessage());
      }
    catch (Exception exception) {
        exception.printStackTrace();
      throw new EJBException(exception);
    }

  }

} // end class SignInBean
```

```
package game.network.server.ejb;

import java.rmi.RemoteException;

import javax.ejb.CreateException;
import javax.ejb.EJBHome;

/**
 * The signin bean is used for users validation, logout and signups.
 */
public interface SignInHome extends EJBHome {

    /**
     * Create a bean instance.
     */
  public SignIn create() throws RemoteException, CreateException;

}
```

```
package game.ship;

import java.util.Random;

/**
 * The AI state class performs actions like movement and
 * firing on a ship according to the events chances defined
 * for this state. It also returns the next state when the ship
 * decides it's time to change state.
 */
public class AIState {

    public static final int AI_TYPE_AGGRESIVE = 1;
    public static final int AI_TYPE_NORMAL = 2;
    public static final int AI_TYPE_COWARD = 3;

    // State chances (each state chances should sum to 1.0)
    public static final float AGGRESIVE_ATTACK_CHANCE = 0.6f;
    public static final float AGGRESIVE_NORMAL_CHANCE = 0.3f;
    public static final float AGGRESIVE_FLEE_CHANCE = 0.1f;

    public static final float COWARD_FLEE_CHANCE = 0.55f;
    public static final float COWARD_NORMAL_CHANCE = 0.30f;
    public static final float COWARD_ATTACK_CHANCE = 0.15f;

    public static final AIState AI_STATE_ATTACK =
        new AIState(AI_TYPE_AGGRESIVE, 0.3f, 0.8f);
    public static final AIState AI_STATE_NORMAL =
        new AIState(AI_TYPE_NORMAL, 0.5f, 0.5f);
    public static final AIState AI_STATE_FLEE =
        new AIState(AI_TYPE_COWARD, 0.7f, 0.2f);

    private float moveChance;
    private float fireChance;
    private int aiType;

    private Random rand;

    /**
     * Construct a new AIState.
     * @param moveChance  Chance for making a movement.
     * @param fireChance  Chance for firing.
     */
    public AIState(int type, float moveChance, float fireChance) {
        this.aiType = type;
        this.moveChance = moveChance;
        this.fireChance = fireChance;

        rand = new Random();
    }

    /**
     * Randomly update the ship according to the events chances.
     * @param ship  Enemy ship to update
     */
    public void update(EnemyShip ship) {

        if (rand.nextFloat() < fireChance) {
            ship.shoot();
        }
```

```java
        if (rand.nextFloat() < moveChance) {
            int direction = (rand.nextFloat() > 0.5) ? -1 : 1;
            ship.setDx(direction*(rand.nextFloat()*ship.getMaxDX()));
        }
        if (rand.nextFloat() < moveChance) {
            int direction = (rand.nextFloat() > 0.5) ? -1 : 1;
            ship.setDy(direction*(rand.nextFloat()*ship.getMaxDX()));
        }
    }

    /**
     * Randomly selects the next AI state. The current state has
     * some affect on the probability of the next state.
     */
    public AIState getNextAIState() {
        AIState nextState = AI_STATE_NORMAL;
        float rand = (float) Math.random();
        switch (aiType) {
            case AI_TYPE_AGGRESIVE:
                if (rand < AGGRESIVE_ATTACK_CHANCE) {
                    nextState = AI_STATE_ATTACK;
                }
                else if (rand < (AGGRESIVE_ATTACK_CHANCE +
AGGRESIVE_NORMAL_CHANCE)) {
                    nextState = AI_STATE_NORMAL;
                }
                else {
                    nextState = AI_STATE_FLEE;
                }
                break;
            case AI_TYPE_COWARD:
                if (rand < COWARD_FLEE_CHANCE) {
                    nextState = AI_STATE_FLEE;
                }
                else if (rand < (AGGRESIVE_ATTACK_CHANCE + COWARD_NORMAL_CHANCE))
{
                    nextState = AI_STATE_NORMAL;
                }
                else {
                    nextState = AI_STATE_ATTACK;
                }
                break;
        }

        return nextState;

    }

}
```

```java
package game.ship;

import game.ship.bonus.*;
import game.ship.weapon.*;
import game.sound.SoundFactory;
import game.util.ResourceManager;

import java.awt.*;

/**
 * The <code>EnemyShip</code> class extends the abstract Ship class
 * and defines common behaviours for enemy ships.
 */
public class EnemyShip extends Ship {

    private final float WEAPON_BONUS_PROBABILITY = 0.07f;
    private final float POWERUP_BONUS_PROBABILITY = 0.8f;

    private final long maxDecisionTime = 10000;

    private long timeInAIState;
    private AIState aiState;  // AI state of the ship

    private long initArmor; // The armor this ship was contructed with

    /**
     * Construct a new enemy ship.
     *
     * @see ShipProperties
     * @see Ship#Ship(int, int, float, float, ShipProperties)
     */
    public EnemyShip(int objectId, int shipType,
            float x, float y, ShipProperties prop) {

        super(objectId, shipType, x, y, prop.maxDX, prop.maxDY,
                prop.image, WeaponFactory.getWeapon(
                        prop.weaponType, prop.weaponLevel,
                        prop.weaponDirection),
                prop.armor, prop.damage,
                prop.hitScoreValue, prop.destroyScoreValue);

        this.initArmor = armor;
        aiState = AIState.AI_STATE_NORMAL;
        timeInAIState = maxDecisionTime;  // force state change

    }

    /**
     * Construct a new enemy ship from a model.
     * @param model ShipModel object
     *
     * @see ShipModel
     * @see Ship#Ship(ShipModel)
     */
    public EnemyShip(ShipModel model) {
        this(model.objectId, model.shipType, model.x, model.y,
                ShipProperties.getShipProperties(model.shipType));
    }

    /**
     * Overrides the render method. Call the super method and
     * add power left of the ship in percentage.
     *
     * @see Ship#render(Graphics)
     */
    public void render(Graphics g) {
        super.render(g);

        if (isActive()) {
            // Draw the armor left to this ship in percents

            float armorLeft = (float)this.armor/this.initArmor;
            int armorPrecent = (int) (armorLeft * 100);

            g.setFont(ResourceManager.getFont(Font.BOLD, 10));

            // The color becomes more reddish when the ship looses armor
            g.setColor(new Color(1-armorLeft, armorLeft, 0.0f));


            g.drawString(armorPrecent+"%",
                    (int)Math.round(this.getX()),
                    (int)Math.round(this.getY())+10);
        }
    }

    /**
     * Override the update method. Call the super method and adds
     * some random activities (shooting, changing direction, etc.).
     *
     * @see Ship#update(long)
     */
    public void update(long elapsedTime) {

        super.update(elapsedTime);

        timeInAIState += elapsedTime;

        boolean changeState = false;
        if (shipContainer.isController()) {

            long randomTime =  (long)(maxDecisionTime * Math.random());

            changeState = timeInAIState >= randomTime ||
              timeInAIState >= maxDecisionTime;

            // Only the controller machine generate random events
          if (changeState) {
              aiState = aiState.getNextAIState();
              timeInAIState = 0;
              aiState.update(this);
          }
        }

        if (shipContainer.isNetworkGame() && shipContainer.isController()
                && (changeState || timeSinceLastPacket > 2500*Math.random())) {
            // Send state update
            createPacket(shipContainer.getNetworkManager());
        }

    }
    /**
     * Hit this ship with the bullet
     */
    public void hit(Bullet bullet) {
        if (isNormal()) {
            SoundFactory.playSound("hit1.wav");
            super.hit(bullet);

          // Add the score to the hitting player
          if (bullet.getOwner() instanceof PlayerShip){
              long damage = bullet.getDamage();
              long actualDamage = Math.min(armor, damage);
              long score = (armor<=0) ? destroyScoreValue :
                  damageScoreValue*actualDamage;
              PlayerShip ship = (PlayerShip) bullet.getOwner();
              ship.addScore(score);
          }
        }
    }

    /**
     * Hit the ship with a bonus. Enemy ships don't consume bonuses.
     */
    public void hit(Bonus bonus) {
        // Enemy ships don't consume bonuses
    }

    /**
     * Returns the enemy ship state.
     */
    public ShipState getShipState() {
        return new ShipState(x, y, dx, dy, armor, state);
    }

    /**
     * Override the destroy methos. Call the super method and randomly
     * drop a bonus.
     *
     * @see Ship#destroy()
     */
    public void destroy() {
        super.destroy();

        if (shipContainer.isController()) {
            // Only the controller generates random events
            Bonus bonus = null;
            double random = Math.random();
            if (random < WEAPON_BONUS_PROBABILITY) {
                bonus = new WeaponUpgrade(getCenterX(),
                        getCenterY(), WeaponFactory.getRandomWeaponType());

            } else if (random < POWERUP_BONUS_PROBABILITY) {
                int powerBonus = Math.round(initArmor * 0.05f);
                bonus = new PowerUp(getCenterX(), getCenterY(), powerBonus);
            }

            if (bonus != null) {
                // Add the bonus and send to the network player
                // if in network game
                getShipContainer().addBonus(bonus);
                if (getShipContainer().isNetworkGame()) {
                    bonus.createPacket(getShipContainer().getNetworkManager());
                }
            }
        }

        // If it's a network game send the ship state to
        // make sure it is destroyed in the other player's world
        if (getShipContainer().isNetworkGame()) {
            createPacket(getShipContainer().getNetworkManager());
        }
    }

}
```

```java
package game.ship;

/**
 * Any movable object in the game should implement the movable
 * interface (the base Sprite class implements this interface).
 */
public interface Movable {

    /**
     * Update the position of the movable object according to the
     * time passed.
     * @param elapsedTime Time elapsed since the last update
     * (in milliseconds).
     */
    public void updatePosition(long elapsedTime);

}
```

```java
package game.ship;

import game.network.packet.Packet;
import game.network.packet.ShipPacket;
import game.ship.bonus.*;
import game.ship.weapon.*;
import game.sound.SoundFactory;
import game.util.Logger;

import java.awt.*;
import java.util.Collection;

/**
 * The <code>PlayerShip</code> class extends the abstract Ship class
 * and defines special behaviours for the player ship.
 */
public class PlayerShip extends Ship {

    private final long MAX_TIME_BETWEEN_PACKETS = 1000;

    private long score;     // Score the player made
    private boolean vulnerable = true;  // True if the player is vulnerable

    /**
     * Construct a new player ship.
     * @see Ship#Ship(int, int, float, float, ShipProperties)
     */
    public PlayerShip(int objectId, int shipType,
            float x, float y, ShipProperties prop) {

        super(objectId, shipType, x, y, prop.maxDX, prop.maxDY,
                prop.image, WeaponFactory.getWeapon(
                        prop.weaponType, prop.weaponLevel,
                        prop.weaponDirection),
                prop.armor, prop.damage,
                prop.hitScoreValue, prop.destroyScoreValue);

        if (Logger.isInvulnerable()) {
            vulnerable = false;
        }

    }

    /**
     * Override the update method. Call the super method and send
     * packet if necessary.
     *
     * @see Ship#update(long)
     */
    public void update(long elapsedTime) {

        super.update(elapsedTime);

        if (shipContainer.isNetworkGame() &&
                timeSinceLastPacket > MAX_TIME_BETWEEN_PACKETS) {
            // Send state update
            createPacket(shipContainer.getNetworkManager());
        }

    }
    /**
     * Override the render methos. Call the super method and
     * render some more data.
     */
    public void render(Graphics g){

        if (isActive()) {
          super.render(g);

          // If not vulnerable draw a bounding sphere
          if (!vulnerable) {
              Graphics2D g2 = (Graphics2D)g;

              g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                      RenderingHints.VALUE_ANTIALIAS_ON);

              g.setColor(Color.GREEN);
              g.drawOval((int) Math.round(getX()) - 3,
                      (int) Math.round(getY()) - 10, getWidth() + 6,
                      getHeight() + 20);
          }
        }

    }

    /**
     * Process ship-to-ship collisions only if this ship
     * is vulnerable.
     */
    public void processCollisions(Collection targets) {

        if (vulnerable) {
            super.processCollisions(targets);
        }
    }

    /**
     * Returns true if the player ship is vulnerable.
     * @return True if the player ship is vulnerable.
     */
    public boolean isVulnerable() {
        return vulnerable;
    }

    /**
     * Sets the vulnerable state of the ship.
     * @param vulnerable  True if ship vulnerable.
     */
    public void setVulnerable(boolean vulnerable) {
        this.vulnerable = vulnerable;
    }

    /**
     * Add score to the player.
     * @param value Score to add.
     */
    public void addScore(long value) {
        score += value;
    }
```

```java
    /**
     * Returns the player score.
     * @return The player's score.
     */
    public long getScore() {
        return this.score;
    }

    /**
     * Returns the player's ship state.
     */
    public ShipState getShipState() {
        return new PlayerShipState(x, y, dx, dy, armor, state, score);
    }

    /**
     * Hit the player ship if the ship is vulnerable.
     */
    public void hit(long damage) {
        if (vulnerable) {
            super.hit(damage);
        }
    }

    /**
     * Hit only if vulnerable.
     */
    public void hit(Bullet bullet) {
        if (vulnerable && isNormal()) {
            SoundFactory.playSound("playerHit.wav");
            super.hit(bullet);
        }
    }

    /**
     * Hit the player ship with a bonus.
     */
    public void hit(Bonus bonus) {
        if (isNormal()) {
            if (bonus instanceof PowerUp) {
                PowerUp powerUp = (PowerUp)bonus;

                SoundFactory.playSound("bonus.wav");

                long power = powerUp.getPowerUp();
                armor += power;

            } else if (bonus instanceof WeaponUpgrade) {
                WeaponUpgrade weaponUpgrade = (WeaponUpgrade)bonus;

                SoundFactory.playSound("weapon_bonus.wav");

                int weaponType = weaponUpgrade.getWeaponType();

                if (weaponType == this.weapon.getWeaponType()) {
                    // Same weapon, increase the level by 1
                    weapon.upgradeWeapon();
                } else {
                    // Different weapon, just replace current weapon
                    Weapon newWeapon = WeaponFactory.getWeapon(weaponType,
                            weapon.getWeaponLevel(), Weapon.DIRECTION_UP);
                    newWeapon.setOwner(this);

                    this.setWeapon(newWeapon);
                }
            }
        }
    } // end method hit

    /**
     * Handle incoming packet.
     */
    public void handlePacket(Packet packet) {

        super.handlePacket(packet);

        if (packet instanceof ShipPacket) {
            ShipPacket shipPacket = (ShipPacket)packet;
            PlayerShipState shipState =
                (PlayerShipState)shipPacket.getShipState();
            this.score = shipState.score;
        }

    }

    /**
     * This method is called when the <code>PlayerShipManager</code>
     * wants to force the player ship to send packet on the next
     * update.
     */
    public void forcePacket() {
        // Set the time since last packet to the max time
        // to force the sending of a ship packet
        timeSinceLastPacket = MAX_TIME_BETWEEN_PACKETS;
    }

}
```

```java
package game.ship;

/**
 * The player ship state adds some more data needed for a
 * player ship.
 */
public class PlayerShipState extends ShipState {

    public long score;  // The player's score

    /**
     * Construct a new ship state.
     * @param x     Current vertical position.
     * @param y     Current horizontal position.
     * @param dx    Current ship's vertical velocity.
     * @param dy    Current ship's horizontal velocity.
     * @param armor   Current ship's armor.
     * @param state   Current ship's state.
     * @param score   The player's score
     */
    public PlayerShipState(float x, float y, float dx, float dy, long armor,
            int state, long score) {
        super(x, y, dx, dy, armor, state);
        this.score = score;

    }

}
```

```java
package game.ship;

import java.awt.Graphics;

/**
 * The active objects in the game should implement this interface.
 * (The base Sprite class implements this interface)
 */
public interface Renderable {

    /**
     * Render the object givet the graphics context.
     * @param g   A <code>Graphics</code> object.
     */
    public void render(Graphics g);

}
```

```java
package game.ship;

import game.network.client.GameNetworkManager;
import game.network.packet.*;
import game.ship.weapon.*;
import game.sound.SoundFactory;

import java.awt.Graphics;
import java.awt.Image;
import java.util.Collection;
import java.util.Iterator;

/**
 * The abstract <code>Ship</code> class is the base class for all the
 * ships in the game/
 */
public abstract class Ship extends Sprite implements Target, PacketHandler {

    private final static int STATE_NORMAL = 0;
    private final static int STATE_EXPLODING = 1;
    private final static int STATE_DESTROYED = 2;

    protected ShipContainer shipContainer;

    /** Time passed since the last packet send */
    protected long timeSinceLastPacket;

    protected int objectId;
    private int shipType;

    /** The ship's armor */
    protected long armor;
    /** The damage the ship causes when colliding with a traget */
    private long damage;

    /** Max vertical and horizontal velocity of the ship */
    protected float maxDX, maxDY;

    protected long damageScoreValue;
    protected long destroyScoreValue;

    protected int state = STATE_NORMAL;

    protected Weapon weapon;
    //private Weapon secondWeapon;

    /**
     * Construct a new ship.
     * @param objectId  Network handler id of the ship
     * @param shipType  Type of the ship as defined in ShipProperties
     * @param x   Vertical location of the ship (from left)
     * @param y   Horizontal location of the ship (from top)
     * @param dx  Max vertical velocity (pixels/sec)
     * @param dy  Max horizontal velocity (pixels/sec)
     * @param image Ship image
     * @param gun Ship main weapon
     * @param armor Ship armor
     * @param damage  Damage the ship cause to other ship when they collide
     * @param hitScoreValue Score the ship gives per 1 damage unit
     * @param destroyScoreValue Score the ship gives when destroyed
     */
    public Ship(int objectId, int shipType,
            float x, float y, float dx, float dy,
            Image image, Weapon gun, long armor, long damage,
            long hitScoreValue, long destroyScoreValue) {

        super(x, y, dx, dy, image);
        this.objectId = objectId;
        this.shipType = shipType;
        this.maxDX = dx;
        this.maxDY = dy;
        this.weapon = gun;
        this.armor = armor;
        this.damage = damage;
        this.damageScoreValue = hitScoreValue;
        this.destroyScoreValue = destroyScoreValue;

        // Set this ship as the owner of the weapon
        this.weapon.setOwner(this);

        this.timeSinceLastPacket = 0;
    }


    /**
     * Costruct a new ship from ship type.
     * @param objectId  Network handler id of the ship
     * @param shipType  Type of the ship as defined in ShipProperties
     * @param x     Vertical location of the ship (from left)
     * @param y     Horizontal location of the ship (from top)
     * @param prop    Properties of the ship
     */
    public Ship(int objectId, int shipType, float x, float y,
            ShipProperties prop) {

        this(objectId, shipType, x, y, prop.maxDX, prop.maxDY,
                prop.image, WeaponFactory.getWeapon(prop.weaponType,
                        prop.weaponLevel, prop.weaponDirection) ,
                prop.armor, prop.damage,
                prop.hitScoreValue, prop.destroyScoreValue);

    }


    /**
     * Construct a new ship from a <code>ShipModel</code>.
     * @param model ShipModel with the ship details
     * @see game.ship.ShipModel
     */
    public Ship(ShipModel model) {
        this(model.objectId, model.shipType, model.x, model.y,
                ShipProperties.getShipProperties(model.shipType));
    }


    /**
     * Calls the main weapon to fire.
     */
    public void shoot() {
        weapon.fire(getCenterX(), getY());
    }
```

```java
    /**
     * Check if this ship collides with one of the targets(ships).
     * If so hit it and with <code>damage</code>.
     * Note that this method is for ship-to-ship collision only.
     * @param targets Collection of target ships.
     */
    public void processCollisions(Collection targets) {

        if (!active) {
            return;
        }

        int x0 = (int)Math.round(this.getX());
        int y0 = (int)Math.round(this.getY());
        int x1 = x0 + this.getWidth();
        int y1 = y0 + this.getHeight();

        Iterator targetsItr = targets.iterator();
        while (targetsItr.hasNext()) {
            Target target = (Target) targetsItr.next();
            if (target.isCollision(x0, y0, x1, y1)) {
                target.hit(this.getDamage());
            }
        }

    }

    /**
     * Returns true if this ship collide with the rectangle
     * (x0, y0), (x1, y1) (top-left and bottom-right respectively)
     */
    public boolean isCollision(int x0, int y0, int x1, int y1) {

        if (state == STATE_DESTROYED) {
            return false;
        }
        else {

          // get the pixel location of this ship
          int s2x = (int)Math.round(this.getX());
          int s2y = (int)Math.round(this.getY());
          int s2x1 = s2x + this.getWidth();
          int s2y1 = s2y + this.getHeight();

          // check if the boundaries intersect
          return ( x0 < s2x1 &&
                   s2x < x1  &&
                   y0 < s2y1 &&
                   s2y < y1);
        }
    }

    /**
     * Called by a game object to hit this ship.
     * @param damage Amount of damage the hit cause
     */
    public void hit(long damage) {
        if (state == STATE_NORMAL) {

            SoundFactory.playSound("hit1.wav");
            long actualDamage = Math.min(armor, damage);
            armor -= actualDamage;
          if (armor == 0) {
              destroy();
          }
        }
    }

    /**
     * Called by a bullet when it hits this ship.
     * If the owner of the bullet is a player ship we add
     * score to the owner ship.
     */
    public void hit(Bullet bullet) {
        if (state == STATE_NORMAL) {
            long damage = bullet.getDamage();
            long actualDamage = Math.min(armor, damage);
            armor -= actualDamage;
          if (armor == 0) {
              destroy();
          }
        }
    }

    /**
     * Updates the ship's state.
     * @param elapsedTime Time passed since the last update.
     */
    public void update(long elapsedTime) {
        if (isActive()) {
            timeSinceLastPacket += elapsedTime;
          updatePosition(elapsedTime);
        }
    }

    /**
     * Render the ship.
     */
    public void render(Graphics g) {
        if (isActive()) {
          super.render(g);
        }
    }

    /**
     * Destroy this ship.
     */
    public void destroy() {
        SoundFactory.playSound("explode1.wav");
        this.setActive(false);
        this.state = STATE_DESTROYED;
    }

    /**
     * Sets the main weapon of the ship.
     * @param weapon  New main weapon
     */
    protected void setWeapon(Weapon weapon) {
        this.weapon = weapon;
    }
    /**
     * Returns the damage caused by this ship when it collides with
     * other game objects.
     * @return  Damage caused by this ship.
     */
    public long getDamage() {
        return this.damage;
    }

    /**
     * Returns the ship armot (hit points).
     * @return The ship armor.
     */
    public long getArmor() {
        return this.armor;
    }

    /**
     * Returns the max vertical velocity of the ship.
     * @return Max vertical velocity of the ship.
     */
    public float getMaxDX() {
        return this.maxDX;
    }

    /**
     * Returns the max horizontal velocity of the ship.
     * @return Max horizontal velocity of the ship.
     */
    public float getMaxDY() {
        return this.maxDY;
    }

    public void setShipContainer(ShipContainer container) {
        this.shipContainer = container;
    }

    /**
     * Returns true if this ship state is in normal state (can fight).
     * @return True if this ship is in normal state.
     */
    public boolean isNormal() {
        return state == STATE_NORMAL;
    }

    /**
     * Returns true if this ship state is exploding.
     * @return True if this ship is exploding.
     */
    public boolean isExploding() {
        return state == STATE_EXPLODING;
    }

    /**
     * Returns true if this ship state is destroyed.
     * @return True if this ship is destroyed.
     */
    public boolean isDestroyed() {
        return state == STATE_DESTROYED;
    }
    /**
     * Returns <code>ShipModel</code> object of this ship.
     * @return  ShipModel object of this ship
     * @see game.ship.ShipModel
     */
    public ShipModel getShipModel() {
        return new ShipModel(objectId, shipType,
                getX(), getY(), getDx(), getDy());
    }

    /**
     * Returns <code>ShipState</code> object of this ship to send over
     * the network. This method is abstract since the player's ship
     * and enemy ships send different ShipState objects.
     * @return  ShipState object of this ship
     * @see game.ship.ShipState
     */
    public abstract ShipState getShipState();

    /**
     * Handle incoming packet.
     * Called by the <code>ShipContainer</code> according to the ship
     * network id.
     */
    public void handlePacket(Packet packet) {

        if (packet instanceof ShipPacket) {
            ShipPacket shipPacket = (ShipPacket)packet;
            ShipState shipState = shipPacket.getShipState();

            setX(shipState.x);
            setY(shipState.y);
            setDx(shipState.dx);
            setDy(shipState.dy);
            armor = shipState.armor;
            state = shipState.state;

            packet.setConsumed(true);
        }

    }

    /**
     * Generates and sends a packet with the details of this ship.
     */
    public void createPacket(GameNetworkManager netManager) {

        ShipState shipState = getShipState();

        Packet shipPacket = new ShipPacket(netManager.getSenderId(),
                netManager.getReceiverId(), getHandlerId(), shipState);

        netManager.sendPacket(shipPacket);

        timeSinceLastPacket = 0;
    }

    /**
     * Returns the network handler id of this ship.
     * @return  Network handler id
     */
    public int getHandlerId() {
        return this.objectId;
    }

    /**
     * Returns the <code>ShipContainer</code> object containing this ship.
     * @return ShipContainer object
     */
    public ShipContainer getShipContainer() {
        return shipContainer;
    }
}
```

```java
package game.ship;

import game.ship.bonus.Bonus;
import game.ship.weapon.Bullet;
import game.network.client.GameNetworkManager;

/**
 * The <code>ShipContainer</code> defines the methods that an
 * object containing and manages ships should implement.
 * Through this interface the ships can cummunicate with other
 * game objects.
 */
public interface ShipContainer {

    /**
     * Adds a new ship to the ship container.
     * @param ship  Ship to add.
     */
    public void addShip(Ship ship);

    /**
     * Adds a shot to the ship container shot collection.
     * @param shot  Shot to add.
     */
    public void addShot(Bullet shot);

    /**
     * Adds a bonus to the ship container bonuses.
     * @param bonus Bonus to add.
     */
    public void addBonus(Bonus bonus);

    /**
     * Returns the network manager of the game.
     * @return  Game network manager.
     */
    public GameNetworkManager getNetworkManager();

    /**
     * Returns the network handler id of the ship container.
     * @return Network handler id of the ship container.
     */
    public int getHandlerId();

    /**
     * Returns true if the game is a network game.
     * @return  True if in network game.
     */
    public boolean isNetworkGame();

    /**
     * Returns true if this machine is the controller machine.
     * @return True if this machine is the controller machine.
     */
    public boolean isController();

}
```

```java
package game.ship;

import java.io.Serializable;

/**
 * This class represents a model of a ship to be sent to the
 * network player when the ship is created (start of new level).
 */
public class ShipModel implements Serializable {

    public int objectId, shipType;
    public float x, y, dx, dy;

    /**
     * Construct a new ship model.
     * @param objectId  Id of the ship
     * @param shipType  Type of the ship
     * @param x   Vertical location of the ship (from left)
     * @param y   Horizontal location of the ship (from top)
     * @param dx  Max vertical velocity (pixels/sec)
     * @param dy  Max horizontal velocity (pixels/sec)
     */
    public ShipModel (int objectId, int shipType,
            float x, float y, float dx, float dy) {

        this.objectId = objectId;
        this.shipType = shipType;
        this.x = x;
        this.y = y;
        this.dx = dx;
        this.dy = dy;
    }

}
```

```java
package game.ship;

import game.GameConstants;
import game.ship.weapon.Weapon;
import game.ship.weapon.WeaponFactory;
import game.util.ResourceManager;

import java.awt.Image;
import java.util.HashMap;
import java.util.Map;

/**
 * The <code>ShipProperties</code> class holds the properties of all the
 * ships in the game.
 */
public class ShipProperties {

    // Various ship types
    public static final int SHIP_TYPE_1 = 1;
    public static final int SHIP_TYPE_2 = 2;
    public static final int SHIP_TYPE_3 = 3;
    public static final int SHIP_TYPE_4 = 4;
    public static final int SHIP_TYPE_5 = 5;
    public static final int SHIP_TYPE_6 = 6;
    public static final int SHIP_TYPE_7 = 7;
    public static final int SHIP_TYPE_8 = 8;

    public static final int ROBO_SHIP_TYPE_1 = 21;
    public static final int ROBO_SHIP_TYPE_2 = 22;
    public static final int ROBO_SHIP_TYPE_3 = 23;
    public static final int ROBO_SHIP_TYPE_4 = 24;
    public static final int ROBO_SHIP_TYPE_5 = 25;
    public static final int ROBO_SHIP_TYPE_6 = 26;
    public static final int ROBO_SHIP_TYPE_7 = 27;

    public static final int CARDROM_SHIP_TYPE_1 = 41;
    public static final int CARDROM_SHIP_TYPE_2 = 42;
    public static final int CARDROM_SHIP_TYPE_3 = 43;
    public static final int CARDROM_SHIP_TYPE_4 = 44;
    public static final int CARDROM_SHIP_TYPE_5 = 45;
    public static final int CARDROM_SHIP_TYPE_6 = 46;

    public static final int COLOR_SHIP_TYPE_1 = 61;
    public static final int COLOR_SHIP_TYPE_2 = 62;
    public static final int COLOR_SHIP_TYPE_3 = 63;
    public static final int COLOR_SHIP_TYPE_4 = 64;

    public static final int PLAYER_SHIP_TYPE_1 = 100;
    public static final int PLAYER_SHIP_TYPE_2 = 101;

    public static final int BOSS_SHIP_TYPE_1 = 201;
    public static final int BOSS_SHIP_TYPE_2 = 202;
    public static final int BOSS_SHIP_TYPE_3 = 203;
    public static final int BOSS_SHIP_TYPE_4 = 204;
    public static final int BOSS_SHIP_TYPE_5 = 205;

    public static Map shipsProperties = initShipsProperties();

    public float maxDX, maxDY;
    public long armor;
    public long damage;
    public long hitScoreValue;
    public long destroyScoreValue;
    public Image image;
    public String imageName;
    public int weaponType;
    public int weaponLevel;
    public int weaponDirection;

    /**
     * Initialize the <code>ShipProperties</code> for the various ships.
     * @return  Map whit the ship type as key and ShipProperties as value.
     */
    private static Map initShipsProperties() {

        Map shipsProperties = new HashMap();

        final ShipProperties PlayerShipType1 = new ShipProperties(0.25f, 0.25f,
                20, 1, 10, 500, "player1.png", WeaponFactory.TYPE_FIRE_CANNON,
                1, Weapon.DIRECTION_UP);

        shipsProperties.put(new Integer(PLAYER_SHIP_TYPE_1), PlayerShipType1);

        final ShipProperties PlayerShipType2 = new ShipProperties(0.25f, 0.25f,
                20, 1, 10, 500, "player2.png", WeaponFactory.TYPE_FIRE_CANNON,
                1, Weapon.DIRECTION_UP);

        shipsProperties.put(new Integer(PLAYER_SHIP_TYPE_2), PlayerShipType2);

        final ShipProperties ShipType1 = new ShipProperties(0.07f, 0.07f,
                20, 1, 10, 500, "ZamsAirspeeder.png",
                WeaponFactory.TYPE_LASER_CANNON, 1, Weapon.DIRECTION_DOWN);

        shipsProperties.put(new Integer(SHIP_TYPE_1), ShipType1);

        final ShipProperties ShipType2 = new ShipProperties(0.09f, 0.09f,
                30, 1, 15, 1000, "RepublicGunship.png",
                WeaponFactory.TYPE_LASER_CANNON, 1, Weapon.DIRECTION_DOWN);

        shipsProperties.put(new Integer(SHIP_TYPE_2), ShipType2);

        final ShipProperties ShipType3 = new ShipProperties(0.11f, 0.11f,
                40, 2, 20, 2500, "RepublicAssaultShip.png",
                WeaponFactory.TYPE_LASER_CANNON, 1, Weapon.DIRECTION_DOWN);

        shipsProperties.put(new Integer(SHIP_TYPE_3), ShipType3);

        final ShipProperties ShipType4 = new ShipProperties(0.12f, 0.12f,
                50, 2, 30, 4500, "bajoran2.png",
                WeaponFactory.TYPE_LASER_CANNON, 2, Weapon.DIRECTION_DOWN);

        shipsProperties.put(new Integer(SHIP_TYPE_4), ShipType4);

        final ShipProperties BossShipType1 = new ShipProperties(0.07f, 0.07f,
                500, 2, 15, 50000, "boss1.png",
                WeaponFactory.TYPE_BALL_CANNON, 2, Weapon.DIRECTION_DOWN);

        shipsProperties.put(new Integer(BOSS_SHIP_TYPE_1), BossShipType1);

        final ShipProperties RoboShipType1 = new ShipProperties(0.05f, 0.05f,
                25, 2, 12, 550, "robo1.png",
                WeaponFactory.TYPE_LASER_CANNON, 1, Weapon.DIRECTION_DOWN);
        shipsProperties.put(new Integer(ROBO_SHIP_TYPE_1), RoboShipType1);

        final ShipProperties RoboShipType2 = new ShipProperties(0.05f, 0.05f,
                35, 2, 15, 550, "robo2.png",
                WeaponFactory.TYPE_LASER_CANNON, 1, Weapon.DIRECTION_DOWN);

        shipsProperties.put(new Integer(ROBO_SHIP_TYPE_2), RoboShipType2);

        final ShipProperties RoboShipType3 = new ShipProperties(0.11f, 0.11f,
                20, 1, 30, 800, "robo3.png",
                WeaponFactory.TYPE_LASER_CANNON, 2, Weapon.DIRECTION_DOWN);

        shipsProperties.put(new Integer(ROBO_SHIP_TYPE_3), RoboShipType3);

        final ShipProperties RoboShipType4 = new ShipProperties(0.07f, 0.07f,
                65, 2, 20, 1000, "robo4.png",
                WeaponFactory.TYPE_LASER_CANNON, 2, Weapon.DIRECTION_DOWN);

        shipsProperties.put(new Integer(ROBO_SHIP_TYPE_4), RoboShipType4);

        final ShipProperties RoboShipType5 = new ShipProperties(0.07f, 0.07f,
                75, 2, 15, 1500, "robo5.png",
                WeaponFactory.TYPE_LASER_CANNON, 3, Weapon.DIRECTION_DOWN);

        shipsProperties.put(new Integer(ROBO_SHIP_TYPE_5), RoboShipType5);

        final ShipProperties RoboShipType6 = new ShipProperties(0.08f, 0.08f,
                110, 3, 11, 1500, "robo6.png",
                WeaponFactory.TYPE_LASER_CANNON, 2, Weapon.DIRECTION_DOWN);

        shipsProperties.put(new Integer(ROBO_SHIP_TYPE_6), RoboShipType6);

        final ShipProperties RoboShipType7 = new ShipProperties(0.06f, 0.06f,
                150, 3, 20, 2500, "robo7.png",
                WeaponFactory.TYPE_LASER_CANNON, 4, Weapon.DIRECTION_DOWN);

        shipsProperties.put(new Integer(ROBO_SHIP_TYPE_7), RoboShipType7);

        final ShipProperties BossShipType2 = new ShipProperties(0.08f, 0.08f,
                750, 3, 15, 60000, "boss2.png",
                WeaponFactory.TYPE_BALL_CANNON, 4, Weapon.DIRECTION_DOWN);

        shipsProperties.put(new Integer(BOSS_SHIP_TYPE_2), BossShipType2);

        final ShipProperties BossShipType3 = new ShipProperties(0.09f, 0.09f,
                1550, 3, 20, 75000, "boss3.png",
                WeaponFactory.TYPE_BALL_CANNON, 5, Weapon.DIRECTION_DOWN);

        shipsProperties.put(new Integer(BOSS_SHIP_TYPE_3), BossShipType3);

        final ShipProperties CardromShipType1 = new ShipProperties(0.1f, 0.1f,
                90, 2, 15, 1000, "cardrom1.png",
                WeaponFactory.TYPE_LASER_CANNON, 4, Weapon.DIRECTION_DOWN);

        shipsProperties.put(new Integer(CARDROM_SHIP_TYPE_1), CardromShipType1);

        final ShipProperties CardromShipType2 = new ShipProperties(0.08f, 0.08f,
                100, 2, 20, 1200, "cardrom2.png",
                WeaponFactory.TYPE_LASER_CANNON, 5, Weapon.DIRECTION_DOWN);
        shipsProperties.put(new Integer(CARDROM_SHIP_TYPE_2), CardromShipType2);

        final ShipProperties CardromShipType3 = new ShipProperties(0.09f, 0.09f,
                120, 2, 20, 1500, "cardrom3.png",
                WeaponFactory.TYPE_LASER_CANNON, 5, Weapon.DIRECTION_DOWN);

        shipsProperties.put(new Integer(CARDROM_SHIP_TYPE_3), CardromShipType3);

        final ShipProperties CardromShipType4 = new ShipProperties(0.1f, 0.1f,
                200, 2, 15, 2000, "cardrom4.png",
                WeaponFactory.TYPE_LASER_CANNON, 7, Weapon.DIRECTION_DOWN);

        shipsProperties.put(new Integer(CARDROM_SHIP_TYPE_4), CardromShipType4);

        final ShipProperties CardromShipType5 = new ShipProperties(0.16f, 0.16f,
                60, 2, 25, 1500, "cardrom5.png",
                WeaponFactory.TYPE_LASER_CANNON, 5, Weapon.DIRECTION_DOWN);

        shipsProperties.put(new Integer(CARDROM_SHIP_TYPE_5), CardromShipType5);

        final ShipProperties BossShipType4 = new ShipProperties(0.15f, 0.15f,
                2500, 2, 10, 100000, "boss4.png",
                WeaponFactory.TYPE_BALL_CANNON, 7, Weapon.DIRECTION_DOWN);

        shipsProperties.put(new Integer(BOSS_SHIP_TYPE_4), BossShipType4);

        final ShipProperties ColorShipType1 = new ShipProperties(0.17f, 0.17f,
                120, 1, 15, 2500, "color1.png",
                WeaponFactory.TYPE_LASER_CANNON, 5, Weapon.DIRECTION_DOWN);

        shipsProperties.put(new Integer(COLOR_SHIP_TYPE_1), ColorShipType1);

        final ShipProperties ColorShipType2 = new ShipProperties(0.1f, 0.1f,
                220, 2, 20, 3500, "color2.png",
                WeaponFactory.TYPE_LASER_CANNON, 6, Weapon.DIRECTION_DOWN);

        shipsProperties.put(new Integer(COLOR_SHIP_TYPE_2), ColorShipType2);

        final ShipProperties ColorShipType3 = new ShipProperties(0.1f, 0.1f,
                250, 2, 20, 4400, "color3.png",
                WeaponFactory.TYPE_LASER_CANNON, 7, Weapon.DIRECTION_DOWN);

        shipsProperties.put(new Integer(COLOR_SHIP_TYPE_3), ColorShipType3);

        final ShipProperties ColorShipType4 = new ShipProperties(0.2f, 0.2f,
                530, 2, 20, 10000, "color4.png",
                WeaponFactory.TYPE_LASER_CANNON, 8, Weapon.DIRECTION_DOWN);

        shipsProperties.put(new Integer(COLOR_SHIP_TYPE_4), ColorShipType4);

        final ShipProperties BossShipType5 = new ShipProperties(0.16f, 0.16f,
                5000, 2, 10, 200000, "boss5.png",
                WeaponFactory.TYPE_BALL_CANNON, 10, Weapon.DIRECTION_DOWN);

        shipsProperties.put(new Integer(BOSS_SHIP_TYPE_5), BossShipType5);

        return shipsProperties;

    }

    /**
```

```
     * Private constructor for the various ship properties.
     * @param maxDX   Max vertical velocity of the ship (pixels/sec)
     * @param maxDY   Max horizontal velocity of the ship (pixels/sec)
     * @param armor   Armor of the ship (damage it can take)
     * @param damage  Damage the ship cause when it collide with objects
     * @param hitScoreValue   Score the ship gives per 1 damage unit
     * @param destroyScoreValue Score the ship gives when destroyed
     * @param imageName   Ship image name
     * @param weaponType  Predefined weapon type
     * @param weaponLevel Level of the weapon
     * @param weaponDirection Direction of the weapon
     */
    private ShipProperties(float maxDX, float maxDY, long armor, long damage,
            long hitScoreValue, long destroyScoreValue, String imageName,
            int weaponType, int weaponLevel, int weaponDirection) {

        this.maxDX = maxDX;
        this.maxDY = maxDY;
        this.armor = armor;
        this.damage = damage;
        this.hitScoreValue = hitScoreValue;
        this.destroyScoreValue = destroyScoreValue;
        this.image = ResourceManager.loadImage(
                GameConstants.IMAGES_DIR + imageName);
        this.weaponType = weaponType;
        this.weaponLevel = weaponLevel;
        this.weaponDirection = weaponDirection;

    }

    /**
     * Returns the <code>ShipProperties</code> object for the
     * specified type.
     * @param shipType  The ship type.
     */
    public static ShipProperties getShipProperties(int shipType) {

        return (ShipProperties) shipsProperties.get(new Integer(shipType));

    }

}
```

```java
package game.ship;

import java.io.Serializable;

/**
 * The <code>ShipState</code> class holds the state of the ship
 * at the time of this object creation. It is used to send the
 * state to the netwrok player.
 */
public class ShipState implements Serializable {

    public float x, y, dy, dx;
    public long armor;
    public int state;

    /**
     * Construct a new ship state.
     * @param x     Current vertical position.
     * @param y     Current horizontal position.
     * @param dx    Current ship's vertical velocity.
     * @param dy    Current ship's horizontal velocity.
     * @param armor   Current ship's armor.
     * @param state   Current ship's state.
     */
    public ShipState(float x, float y, float dx, float dy,
            long armor, int state) {

        this.x = x;
        this.y = y;
        this.dx = dx;
        this.dy = dy;
        this.armor = armor;
        this.state = state;

    }

}
```

```java
package game.ship;


import java.awt.Graphics;
import java.awt.Image;

/**
 * A <code>Sprite</code> class is the base class for all the movable and
 * renderable game objects in the game (ships, bullets etc.).
 * It contains implementation for general methods needed by all sprites
 * and implements the <code>Renderable</code> and <code>Movable</code>
 * interfaces.
 * This basic sprite uses <code>Image</code> as its rendered object.
 */
public abstract class Sprite implements Movable, Renderable {

    protected float x;     // Vertical position in pixels
    protected float y;     // Horizontal position in pixels
    protected float dx;    // Vertical velocity (pixels/sec)
    protected float dy;    // Horizontal velocity (pixels/sec)

    protected boolean active = true;

    private Image spriteImage;

    /**
     * @param x   Vertical location of the ship (from left)
     * @param y   Horizontal location of the ship (from top)
     * @param dx  Max vertical velocity (pixels/sec)
     * @param dy  Max horizontal velocity (pixels/sec)
     * @param image Sprite's image
     */
    public Sprite(float x, float y, float dx, float dy, Image image) {
        this(x, y, dx, dy);
        this.spriteImage = image;
    }

    /**
     * @param x   Vertical location of the ship (from left)
     * @param y   Horizontal location of the ship (from top)
     * @param dx  Max vertical velocity (pixels/sec)
     * @param dy  Max horizontal velocity (pixels/sec)
     */
    public Sprite(float x, float y, float dx, float dy) {
        this.x = x;
        this.y = y;
        this.dx = dx;
        this.dy = dy;
    }

    /**
     * Updates the sprites position according to the elapsed time.
     * @param elapsedTime Time elapsed since last update in milliseconds
     */
    public void updatePosition(long elapsedTime) {

        x += dx * elapsedTime;
        y += dy * elapsedTime;

    }
    /**
     * Render the image in the current position.
     */
    public void render(Graphics g) {

        g.drawImage(spriteImage, (int)Math.round(x),
                (int)Math.round(y), null);

    }

    /**
     * @return Returns the dx.
     */
    public float getDx() {
        return dx;
    }
    /**
     * @param dx The dx to set.
     */
    public void setDx(float dx) {
        this.dx = dx;
    }
    /**
     * @return Returns the dy.
     */
    public float getDy() {
        return dy;
    }
    /**
     * @param dy The dy to set.
     */
    public void setDy(float dy) {
        this.dy = dy;
    }
    /**
     * @return Returns the x.
     */
    public float getX() {
        return x;
    }
    /**
     * @param x The x to set.
     */
    public void setX(float x) {
        this.x = x;
    }
    /**
     * @return Returns the y.
     */
    public float getY() {
        return y;
    }
    /**
     * @param y The y to set.
     */
    public void setY(float y) {
        this.y = y;
    }
    /**
     * @return Returns the sprite width
     */
    public int getWidth() {
        return spriteImage.getWidth(null);
    }
    /**
     * @return Returns the sprite height
     */
    public int getHeight() {
        return spriteImage.getHeight(null);
    }
    public float getCenterX() {
        return x + (float)(getWidth() / 2);
    }
    public float getCenterY() {
        return y + (float)(getHeight() / 2) ;
    }
    /**
     * @param active
     */
    public void setActive(boolean active) {
        this.active = active;
    }
    /**
     * @return True if the sprite is active
     */
    public boolean isActive() {
        return active;
    }
    /**
     * @return Returns the spriteImage.
     */
    public Image getSpriteImage() {
        return spriteImage;
    }
    /**
     * @param spriteImage The spriteImage to set.
     */
    public void setSpriteImage(Image spriteImage) {
        this.spriteImage = spriteImage;
    }
}
```

```java
package game.ship;

import game.ship.bonus.Bonus;
import game.ship.weapon.Bullet;

/**
 * A <code>Target</code> is an object that can collide with
 * various objects in the game.
 */
public interface Target {

    /**
     * Returns true if this objects collides with the rectangle
     * represented by the point.
     * @param x0  Top left x position
     * @param y0  Top left y position
     * @param x1  Botom right x position
     * @param y1  Botom right y position
     * @return True if this objects collides with the rectangle
     * represented by the point
     */
    public boolean isCollision(int x0, int y0, int x1, int y1);

    /**
     * Hit the target and cause some damage.
     * @param damage  Damage to cause.
     */
    public void hit(long damage);

    /**
     * Hit the target with the bullet.
     * @param bullet  Bullet object.
     */
    public void hit(Bullet bullet);

    /**
     * Hit the target with the bonus.
     * @param bonus Bonus object.
     */
    public void hit(Bonus bonus);

}
```

```java
package game.ship.bonus;

import game.GameConstants;
import game.network.packet.Packet;
import game.network.packet.PacketHandler;
import game.ship.Sprite;
import game.ship.Target;

import java.util.Collection;
import java.util.Iterator;

/**
 * The abstract Bonus class represents a bonus dropped by enemy ship.
 */
public abstract class Bonus extends Sprite implements PacketHandler {

    private boolean hit = false;  // Is this bonus hit a ship
    private static final float dx = 0.0f; // Vertical velocity
    private static final float dy = 0.2f; // Hotizontal velocity

    public Bonus(float x, float y) {

        super(x, y, dx, dy);
    }

    // Implement some of the PacketHandler methods

    public void handlePacket(Packet packet) {
        // Bonuses don't handle incoming packets
    }

    /**
     * The packet handler for all bonuses is the enemy
     * ships manager object.
     */
    public int getHandlerId() {
        return GameConstants.ENEMY_MANAGER_ID;
    }

    /**
     * Test is this bonus collides with one of the input targets.
     * If so, call the ship's <code>hit</code> method and mark as
     * hit.
     * @param targets Collection of <code>Target</code> objects (player ship).
     */
    public void processCollisions(Collection targets) {
        int x0 = Math.round(this.getX());
        int y0 = Math.round(this.getY());
        int x1 = x0 + this.getWidth();
        int y1 = y0 + this.getHeight();

        Iterator targetsItr = targets.iterator();
        while (targetsItr.hasNext() && !hit) {
            Target target = (Target) targetsItr.next();
            if (target.isCollision(x0, y0, x1, y1)) {
                target.hit(this);
                hit = true;
            }
        }
    }
    /**
     * Returns true if this bonus was hit.
     * @return True if this bonus was hit.
     */
    public boolean isHit() {
        return hit;
    }

}
```

```java
package game.ship.bonus;

import game.GameConstants;
import game.graphic.GraphicsHelper;
import game.network.client.GameNetworkManager;
import game.network.packet.Packet;
import game.network.packet.PowerUpPacket;
import game.util.ResourceManager;

import java.awt.*;


/**
 * The <code>PowerUp</code> class is a <code>Bonus</code> which
 * gives the player more armor.
 */
public class PowerUp extends Bonus {

    private static final String imageName = "red_ball.png";
    private int powerUp;  // How much armor to add

    /**
     * Construct a new power up.
     * @param x   Vertical location
     * @param y   Horizontal location
     * @param powerUp Power to add to the ship
     */
    public PowerUp(float x, float y, int powerUp) {

        super(x, y);

        // Load the image for the button and draw the power
        // up on the image
        Image srcImage = ResourceManager.loadImage(
                GameConstants.IMAGES_DIR + imageName);

    Image image = GraphicsHelper.getCompatibleImage(
            srcImage.getWidth(null), srcImage.getHeight(null),
            Transparency.TRANSLUCENT);

    Graphics2D g = (Graphics2D)image.getGraphics();
        g.drawImage(srcImage, 0, 0, null);
        g.setFont(ResourceManager.getFont(16));
        g.setColor(Color.GREEN);
        GraphicsHelper.drawInMiddle(g, image, powerUp+"");
        g.dispose();

        this.setSpriteImage(image);

        this.powerUp = powerUp;

    }

    /**
     * Returns the power this bonus gives.
     * @return  The power this bonus gives.
     */
    public int getPowerUp() {
        return this.powerUp;
    }
    /**
     * Create a <code>PowerUpPacket</code> to send over the net.
     */
    public void createPacket(GameNetworkManager netManager) {

        // Prepare the PowerUpPacket
        Packet packet = new PowerUpPacket(netManager.getSenderId(),
                netManager.getReceiverId(), GameConstants.ENEMY_MANAGER_ID,
                x, y, powerUp);

        netManager.sendPacket(packet);
    }

}
```

```java
package game.ship.bonus;

import java.awt.Image;

import game.GameConstants;
import game.network.client.GameNetworkManager;
import game.network.packet.Packet;
import game.network.packet.WeaponUpgradePacket;
import game.ship.weapon.WeaponFactory;
import game.util.ResourceManager;

/**
 * The <code>WeaponUpgrade</code> class is a <code>Bonus</code> which
 * gives the player one weapon level upgrade or the opportunity to
 * switch the weapon of the ship.
 */
public class WeaponUpgrade extends Bonus {


    private int weaponType; // Type of the weapon this bonus gives

    /**
     * Construct a new WeaponUpgrade
     * @param x       Vertical location
     * @param y       Horizontal location
     * @param weaponType  Type of the weapon this bonus gives
     */
    public WeaponUpgrade(float x, float y, int weaponType) {

        super(x, y);
        this.weaponType = weaponType;
        Image image = ResourceManager.loadImage(GameConstants.IMAGES_DIR +
                WeaponFactory.getWeaponImageByType(weaponType));
        setSpriteImage(image);

    }


    /**
     * Returns the weapon type this bonus represents.
     * @return The weapon type this bonus represents.
     */
    public int getWeaponType() {
        return this.weaponType;
    }

    /**
     * Create a <code>WeaponUpgradePacket</code> to send over the net.
     */
    public void createPacket(GameNetworkManager netManager) {

        // Prepare the PowerUpPacket
        Packet packet = new WeaponUpgradePacket(netManager.getSenderId(),
                netManager.getReceiverId(), GameConstants.ENEMY_MANAGER_ID,
                x, y, weaponType);

        netManager.sendPacket(packet);
    }

}
```

```java
/*
 * Created on 27/04/2005
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
package game.ship.weapon;

import game.ship.Ship;

/**
 * The <code>AbstractWeapon</code> class implements some of
 * the <code>Weapon</code> interface methods and defines some
 * common attributes for weapons.
 */
public abstract class AbstractWeapon implements Weapon {

    // Type of the weapon
    // (the various types are defined in the WeaponFactory class)
    protected final int weaponType;

    // Firing rate of the weapon
    protected long firingRate;

    protected Ship owner;
    protected int direction;
    protected int weaponLevel;

    /**
     * Abstract weapon constructor
     * @param direction   General horizontal direction of the bullets
     * fired by this weapon.
     * @param weaponLevel Level of the weapon
     * @param weaponType  Type of the weapon
     * @param firingRate  Firing rate of the weapon
     */
    public AbstractWeapon (int direction, int weaponLevel,
            int weaponType, long firingRate) {
        this.direction = direction;
        this.weaponLevel = weaponLevel;
        this.weaponType = weaponType;
        this.firingRate = firingRate;
    }


    /**
     * Sets the ship owning the weapon.
     */
    public void setOwner(Ship owner) {
        this.owner = owner;
    }

    /**
     * Get the ship owning the weapon.
     */
    public Ship getOwner() {
        return this.owner;
    }

    /**
     * Add one to the weapon level.
     */
    public void upgradeWeapon() {
        this.weaponLevel++;
    }

    /**
     * Returns the weapon type.
     */
    public int getWeaponType() {
        return this.weaponType;
    }

    /**
     * Returns the weapon level.
     */
    public int getWeaponLevel() {
        return this.weaponLevel;
    }

}
```

```java
package game.ship.weapon;

import game.ship.Ship;

/**
 * Bullet that looks like a ball.
 */
public class BallBullet extends Bullet {

    private final static String imageName = "red_bullet.png";
    private final static long damage = 6;

    /**
     * @see Bullet#Bullet(Ship, float, float, float, float, String, long)
     */
    public BallBullet(Ship owner, float x, float y, float dx, float dy) {
        super(owner, x, y, dx, dy, imageName, damage);

    }

}
```

```java
package game.ship.weapon;

import game.sound.SoundFactory;

/**
 * The <code>BallCannon</code> weapon fires ball bullets.
 * For now, this weapon is used in the enemy boss ships.
 */
public class BallCannon extends AbstractWeapon {

    private static final long BASE_FIRING_RATE = 1000;
    private long lastFiringTime;

    /**
     * Construct a new weapon.
     * @param direction   Base horizontal direction of the weapon.
     * @param weaponLevel Level of the weapon.
     */
    public BallCannon(int direction, int weaponLevel) {
        super(direction, weaponLevel,
                WeaponFactory.TYPE_FIRE_CANNON, BASE_FIRING_RATE);
        this.lastFiringTime = 0;
    }

    /**
     * Fire a new bullet(s).
     */
    public void fire(float x, float y) {

        long now = System.currentTimeMillis();
        long elapsedTime =  now - lastFiringTime;

        if (elapsedTime >= firingRate) {
            lastFiringTime = now;

            float[][] directions = {
                        {0.0f, -0.2f}, {0.2f, 0.0f},
                        {0.1f, 0.1f}, {0.0f, 0.2f},
                        {-0.1f, 0.1f}, {-0.2f, 0.0f}
                };

            // Small chance to shoot the big red balls
            boolean bigAmmo = Math.random() > 0.9;

            for (int i = 0; i < 6; i++) {
                Bullet shoot;

                if (bigAmmo) {
                    shoot = new BigBallBullet(getOwner(), x, y,
                            directions[i][0], directions[i][1]);
                }
                else {
                    shoot = new BallBullet(getOwner(), x, y,
                        directions[i][0], directions[i][1]);
                }

              if (owner.getShipContainer().isNetworkGame()) {
                  // Create and send packet
                  shoot.createPacket(
                          owner.getShipContainer().getNetworkManager());
              }

              getOwner().getShipContainer().addShot(shoot);

            }

            SoundFactory.playSound("fire_shot.wav");
        }
    } // end method fire

}
```

```java
package game.ship.weapon;

import game.ship.Ship;

/**
 * Bullet that looks like a ball. Very damaging bullet.
 */
public class BigBallBullet extends Bullet {

    private final static String imageName = "red_ball.png";
    private final static long damage = 20;

    /**
     * @see Bullet#Bullet(Ship, float, float, float, float, String, long)
     */
    public BigBallBullet(Ship owner, float x, float y, float dx, float dy) {
        super(owner, x, y, dx, dy, imageName, damage);

    }

}
```

```java
package game.ship.weapon;

import game.ship.Ship;

/**
 * The blue laser beam is a kind of laser bullet.
 */
public class BlueLaser extends Bullet {

    private final static String imageName = "bluelaser.png";
    private final static double DX = 0.3;
    private final static long damage = 10;

    /**
     * @see Bullet#Bullet(Ship, float, float, float, float, String, long)
     */
    public BlueLaser (Ship owner, float x, float y, float dx, float dy) {
        super(owner, x, y, dx, dy, imageName, damage);
    }

}
```

```java
package game.ship.weapon;

import game.GameConstants;
import game.network.client.GameNetworkManager;
import game.network.packet.*;
import game.ship.*;

import java.awt.Image;
import java.util.Collection;
import java.util.Iterator;

import javax.swing.ImageIcon;

/**
 * The abstract Bullet class represents a bullet fired by
 * a ship's weapon.
 * Each bullet must have it's owner, meaning the ship that fired it
 */
public abstract class Bullet extends Sprite implements PacketHandler {

    private Ship owner;     // Ship that fired the bullet
    private String imageName; // Image name for the bullet
    private long damage;    // The damage the bullet cause
    private boolean hit;    // True if this bullet hit some object

    /**
     * Construct a new Bullet
     * @param owner   Reference to the ship that fired the bullet
     * @param x     Middle vertical location of the ship (from left)
     * @param y     Horizontal location of the ship (from top)
     * @param dx    Max vertical velocity (pixels/sec)
     * @param dy    Max horizontal velocity (pixels/sec)
     * @param imageName Name of the bullet image
     * @param damage  Damage the bullet cause
     */
    public Bullet(Ship owner, float x, float y, float dx, float dy,
            String imageName, long damage) {

        super(x, y, dx, dy);
        Image image = new ImageIcon(GameConstants.IMAGES_DIR +
imageName).getImage();
        this.setSpriteImage(image);
        // Center the bullet relative to the initial x value
        setX(x-((float)image.getWidth(null)/2));
        this.damage = damage;
        this.owner = owner;

    }


//    public Bullet(Ship owner, float x, float y, float dx, float dy,
//            long damage) {
//
//        super(x, y, dx, dy);
//        this.damage = damage;
//        this.owner = owner;
//
//    }


//    public Bullet(BulletModel model, Ship owner) {
```

```
//
//        this(owner, model.x, model.y, model.dx, model.dy, model.imageName,
model.damage);
//
////        super(model.x, model.y, model.dx, model.dy);
////        this.damage = model.damage;
////        this.owner = owner;
//
//    }


    /**
     * Check if the bullet collides with any of the targets.
     * If it is hit the target and deliver the damage.
     * @param targets Collection of targets to check.
     */
    public void processCollisions(Collection targets) {
        int x0 = Math.round(this.getX());
        int y0 = Math.round(this.getY());
        int x1 = x0 + this.getWidth();
        int y1 = y0 + this.getHeight();

        Iterator targetsItr = targets.iterator();
        while (targetsItr.hasNext() && !hit) {
            Target target = (Target) targetsItr.next();
            if (target.isCollision(x0, y0, x1, y1)) {
                target.hit(this);
                hit = true;
            }
        }
    }

    /**
     * Returns true if this bullet hit some target.
     * @return  True if this bullet hit some target.
     */
    public boolean isHit() {
        return hit;
    }

    /**
     * Returns the damage this bullet can cause.
     * @return  Bullet's damage
     */
    public long getDamage() {
        return this.damage;
    }

    /**
     * Returns the ship that fired this bullet.
     * @return  Ship that fired this bullet.
     */
    public Ship getOwner() {
        return this.owner;
    }


    //////////////////////////////////
    // PacketHandler implementation //
    //////////////////////////////////
    /**
     * Create and send bullet packet.
     */
    public void createPacket(GameNetworkManager netManager) {
        // Prepare the bullet model
        BulletModel model = new BulletModel(this.getClass(),
                owner.getHandlerId(), x, y, dx, dy, damage);

        // Create the BulletPacket
        Packet packet = new BulletPacket(netManager.getSenderId(),
                netManager.getReceiverId(), owner.getHandlerId(), model);

        netManager.sendPacket(packet);

    }

    public void handlePacket(Packet packet) {
        // Bullet is not handling any packets
    }

    /**
     * Returns the network handler id. The handler of bullet
     * packets is the ship container of the firing ship.
     */
    public int getHandlerId() {
        return owner.getShipContainer().getHandlerId();
    }

}
```

```java
package game.ship.weapon;

import java.io.Serializable;

/**
 * The <code>BulletModel</code> class is used to send a bullet
 * fired in the game details to the network player.
 */
public class BulletModel implements Serializable {

    public Class bulletClass;
    public int ownerId;
    public float x, y, dx, dy;
    public long damage;

    /**
     * Construct a new BulletModel.
     * @param bulletClass Class of the bullet
     * @param ownerId   Id of the ship that fired the bullet
     * @param x       Vertical ocation
     * @param y       Horizontal location
     * @param dx      Vertival velocity
     * @param dy      Horizontal velocity
     * @param damage    Damage this bullet cause
     */
    public BulletModel(Class bulletClass, int ownerId, float x, float y,
            float dx, float dy, long damage) {

        this.bulletClass = bulletClass;
        this.ownerId = ownerId;
        this.x = x;
        this.y = y;
        this.dx = dx;
        this.dy = dy;
        this.damage = damage;
    }

}
```

```java
package game.ship.weapon;

import game.ship.Ship;

/**
 * A bullet that looks like a fireball.
 */
public class FireBullet extends Bullet {

    private final static String imageName = "fireshot.gif";
    private final static double DX = 0.2;
    private final static long damage = 7;

    /**
     * @see Bullet#Bullet(Ship, float, float, float, float, String, long)
     */
    public FireBullet(Ship owner, float x, float y, float dx, float dy) {
        super(owner, x, y, dx, dy, imageName, damage);

    }

}
```

```java
package game.ship.weapon;

import game.sound.SoundFactory;

/**
 * The <code>FireCannon</code> weapon mainly fires fire bullets.
 */
public class FireCannon extends AbstractWeapon {

    private static final long BASE_FIRING_RATE = 1100;
    private long lastFiringTime;

    /**
     * Construct a new weapon.
     * @param direction   Base horizontal direction of the weapon.
     * @param weaponLevel Level of the weapon.
     */
    public FireCannon(int direction, int weaponLevel) {
        super(direction, weaponLevel,
                WeaponFactory.TYPE_FIRE_CANNON, BASE_FIRING_RATE);
        this.lastFiringTime = 0;
        updateFiringRate();
    }

    /**
     * Fire a new bullet(s).
     */
    public void fire(float x, float y) {

        long now = System.currentTimeMillis();
        long elapsedTime =  now - lastFiringTime;

        if (elapsedTime >= firingRate) {
            lastFiringTime = now;

            if (weaponLevel == 1) {
              Bullet shoot = new FireBullet(getOwner(), x, y, 0.0f,
                      direction*0.2f);

              if (owner.getShipContainer().isNetworkGame()) {
                  // Create and send packet
                  shoot.createPacket(
                          owner.getShipContainer().getNetworkManager());
              }

              getOwner().getShipContainer().addShot(shoot);

              SoundFactory.playSound("fire_shot.wav");
            }
            else if (weaponLevel == 2) {
              Bullet shoot1 = new FireBullet(getOwner(), x-6, y, 0.0f,
                      direction*0.2f);

              Bullet shoot2 = new FireBullet(getOwner(), x+6, y, 0.0f,
                      direction*0.2f);

              if (owner.getShipContainer().isNetworkGame()) {
                  // Create and send packets
                  shoot1.createPacket(
                          owner.getShipContainer().getNetworkManager());
```

```
                    shoot2.createPacket(
                            owner.getShipContainer().getNetworkManager());
                }

                getOwner().getShipContainer().addShot(shoot1);
                getOwner().getShipContainer().addShot(shoot2);

                SoundFactory.playSound("fire_shot.wav");
            }
            else if (weaponLevel == 3) {
                Bullet shoot1 = new FireBullet(getOwner(), x-6, y+10, 0.0f,
                        direction*0.2f);

                Bullet shoot2 = new FireBullet(getOwner(), x+6, y+10, 0.0f,
                        direction*0.2f);

                Bullet shoot3 = new FireBullet(getOwner(), x, y, 0.0f,
                        direction*0.2f);

                if (owner.getShipContainer().isNetworkGame()) {
                    // Create and send packets
                    shoot1.createPacket(
                            owner.getShipContainer().getNetworkManager());
                    shoot2.createPacket(
                            owner.getShipContainer().getNetworkManager());
                    shoot3.createPacket(
                            owner.getShipContainer().getNetworkManager());
                }

                getOwner().getShipContainer().addShot(shoot1);
                getOwner().getShipContainer().addShot(shoot2);
                getOwner().getShipContainer().addShot(shoot3);

                SoundFactory.playSound("fire_shot.wav");
            }
            else if (weaponLevel >= 4) {
                Bullet shoot1 = new FireBullet(getOwner(), x-6, y+10, 0.0f,
                        direction*0.2f);

                Bullet shoot2 = new FireBullet(getOwner(), x+6, y+10, 0.0f,
                        direction*0.2f);

                Bullet shoot3 = new FireBullet(getOwner(), x, y, 0.0f,
                        direction*0.2f);

                Bullet shoot4 = new FireBullet(getOwner(), x, y+20, 0.0f,
                        direction*0.2f);

                if (owner.getShipContainer().isNetworkGame()) {
                    // Create and send packets
                    shoot1.createPacket(
                            owner.getShipContainer().getNetworkManager());
                    shoot2.createPacket(
                            owner.getShipContainer().getNetworkManager());
                    shoot3.createPacket(
                            owner.getShipContainer().getNetworkManager());
                    shoot4.createPacket(
                            owner.getShipContainer().getNetworkManager());
                }

                getOwner().getShipContainer().addShot(shoot1);
```

```java
              getOwner().getShipContainer().addShot(shoot2);
              getOwner().getShipContainer().addShot(shoot3);
              getOwner().getShipContainer().addShot(shoot4);

              SoundFactory.playSound("fire_shot.wav");
            }
        }
    } // end method fire

    /**
     * Add one to the weapon level.
     */
    public void upgradeWeapon() {
        super.upgradeWeapon();
        updateFiringRate();

    }

    /**
     * Updates the firing rate of the weapon. Makes
     * the weapon faster as the level increases.
     */
    private void updateFiringRate() {
        if (weaponLevel > 1) {
            // Increase the firing rate
            firingRate = BASE_FIRING_RATE - weaponLevel * 7;
        }
    }

}
```

```java
package game.ship.weapon;

import game.ship.Ship;

/**
 * The laser beam is a laser bullet.
 */
public class LaserBeam extends Bullet {

    private final static String imageName = "laserbeam.png";
    private final static double DX = 0.3;
    private final static long damage = 5;

    /**
     * @see Bullet#Bullet(Ship, float, float, float, float, String, long)
     */
    public LaserBeam (Ship owner, float x, float y, float dx, float dy) {
        super(owner, x, y, dx, dy, imageName, damage);
    }

}
```

```java
package game.ship.weapon;

import game.sound.SoundFactory;

/**
 * The <code>LaserCannon</code> weapon mainly fires lasers.
 */
public class LaserCannon extends AbstractWeapon {

    private static final long BASE_FIRING_RATE = 950;

    private long lastFiringTime;

    /**
     * Construct a new weapon.
     * @param direction   Base horizontal direction of the weapon.
     * @param weaponLevel Level of the weapon.
     */
    public LaserCannon(int direction, int weaponLevel) {
        super(direction, weaponLevel,
                WeaponFactory.TYPE_LASER_CANNON, BASE_FIRING_RATE);
        this.lastFiringTime = 0;
        updateFiringRate();
    }

    /**
     * Fire a new bullet(s).
     */
    public void fire(float x, float y) {

        long now = System.currentTimeMillis();
        long elapsedTime =  now - lastFiringTime;

        if (elapsedTime >= firingRate) {
            lastFiringTime = now;

            if (weaponLevel == 1) {
              // Fire one laser up
              Bullet shoot = new LaserBeam(getOwner(), x, y, 0.0f,
                      direction*0.2f);

              if (owner.getShipContainer().isNetworkGame()) {
                  // Create and send packet
                  shoot.createPacket(
                          owner.getShipContainer().getNetworkManager());
              }

              getOwner().getShipContainer().addShot(shoot);

              SoundFactory.playSound("enemylaser.wav");

            }
            else if (weaponLevel == 2) {
              // Fire two lasers up
              Bullet shoot1 = new LaserBeam(getOwner(), x-6, y, 0.0f,
                      direction*0.2f);

              Bullet shoot2 = new LaserBeam(getOwner(), x+6, y, 0.0f,
                      direction*0.2f);
              if (owner.getShipContainer().isNetworkGame()) {
                  // Create and send packet
                  shoot1.createPacket(
                          owner.getShipContainer().getNetworkManager());
                  shoot2.createPacket(
                          owner.getShipContainer().getNetworkManager());
              }

              getOwner().getShipContainer().addShot(shoot1);
              getOwner().getShipContainer().addShot(shoot2);

              SoundFactory.playAppletClip("enemylaser.wav");
          }
          else if (weaponLevel == 3) {
              // Fire one blue laser up
              Bullet shoot = new BlueLaser(getOwner(), x, y, 0.0f,
                      direction*0.2f);


              if (owner.getShipContainer().isNetworkGame()) {
                  // Create and send packet
                  shoot.createPacket(
                          owner.getShipContainer().getNetworkManager());
              }

              getOwner().getShipContainer().addShot(shoot);

              SoundFactory.playSound("enemylaser.wav");

          }
          else if (weaponLevel == 4) {
                // Fire two diagonal lasers and one up
              Bullet shoot1 = new LaserBeam(getOwner(), x-6, y, 0.05f,
                      direction*0.2f);

              Bullet shoot2 = new LaserBeam(getOwner(), x+6, y, -0.05f,
                      direction*0.2f);

              Bullet shoot3 = new LaserBeam(getOwner(), x, y, 0.0f,
                      direction*0.2f);

              if (owner.getShipContainer().isNetworkGame()) {
                  // Create and send packet
                  shoot1.createPacket(
                          owner.getShipContainer().getNetworkManager());
                  shoot2.createPacket(
                          owner.getShipContainer().getNetworkManager());
                  shoot3.createPacket(
                          owner.getShipContainer().getNetworkManager());
              }

              getOwner().getShipContainer().addShot(shoot1);
              getOwner().getShipContainer().addShot(shoot2);
              getOwner().getShipContainer().addShot(shoot3);

              SoundFactory.playAppletClip("enemylaser.wav");
          }
          else if (weaponLevel == 5) {
              // Fire two blue lasers up
              Bullet shoot1 = new BlueLaser(getOwner(), x-6, y, 0.0f,
                      direction*0.2f);
              Bullet shoot2 = new BlueLaser(getOwner(), x+6, y, 0.0f,
                      direction*0.2f);

              if (owner.getShipContainer().isNetworkGame()) {
                  // Create and send packet
                  shoot1.createPacket(
                          owner.getShipContainer().getNetworkManager());
                  shoot2.createPacket(
                          owner.getShipContainer().getNetworkManager());
              }

              getOwner().getShipContainer().addShot(shoot1);
              getOwner().getShipContainer().addShot(shoot2);

              SoundFactory.playAppletClip("enemylaser.wav");
            }

            else if (weaponLevel >= 6) {
                // Fire two diagonal blue lasers and one up
              Bullet shoot1 = new BlueLaser(getOwner(), x-6, y, 0.05f,
                      direction*0.2f);

              Bullet shoot2 = new BlueLaser(getOwner(), x+6, y, -0.05f,
                      direction*0.2f);

              Bullet shoot3 = new BlueLaser(getOwner(), x, y, 0.0f,
                      direction*0.2f);

              if (owner.getShipContainer().isNetworkGame()) {
                  // Create and send packet
                  shoot1.createPacket(
                          owner.getShipContainer().getNetworkManager());
                  shoot2.createPacket(
                          owner.getShipContainer().getNetworkManager());
                  shoot3.createPacket(
                          owner.getShipContainer().getNetworkManager());
              }

              getOwner().getShipContainer().addShot(shoot1);
              getOwner().getShipContainer().addShot(shoot2);
              getOwner().getShipContainer().addShot(shoot3);

              SoundFactory.playAppletClip("enemylaser.wav");
            }

        }
    } // end method fire

    /**
     * Add one to the weapon level.
     */
    public void upgradeWeapon() {
        super.upgradeWeapon();
        updateFiringRate();

    }

    /**
     * Updates the firing rate of the weapon. Makes
     * the weapon faster as the level increases.
```

```java
     */
    private void updateFiringRate() {
        if (weaponLevel > 1) {
            // Increase the firing rate
            firingRate = BASE_FIRING_RATE - weaponLevel * 15;
        }
    }

}
```

```java
package game.ship.weapon;

import game.ship.Ship;

import java.io.Serializable;

/**
 * All the weapons in the game should implment this interface.
 */
public interface Weapon extends Serializable {

    /** Horizontal direction of the weapon (usually enemies fire down) */
    public static final int DIRECTION_UP = -1;
    public static final int DIRECTION_DOWN = 1;

    /**
     * Sets the owner of the weapon.
     * @param owner Owning ship of the weapon.
     */
    public void setOwner(Ship owner);

    /**
     * Returns the owner of the weapon.
     * @return The owner of the weapon.
     */
    public Ship getOwner();

    /**
     * Fire a bullet (or more) from the weapon
     * @param x   Vertical location to fire from
     * @param y   Horizontal location to fire from
     */
    public void fire(float x, float y);

    /**
     * Returns the type of the weapon (the weapon types
     * are defined in the <code>WeaponFactory</code> class.
     * @return The type of the weapon
     */
    public int getWeaponType();

    /**
     * Returns the level of the weapon. The more advanced is the level
     * the more lethal the weapon's shots.
     * @return  The level of the weapon.
     */
    public int getWeaponLevel();

    /**
     * This method is called when the owning ship upgrades it's
     * weapon (received bonus). The weapon should advance to the
     * next level of add damage to the bullets.
     */
    public void upgradeWeapon();

}
```

```java
package game.ship.weapon;

import game.ship.Ship;

/**
 * The <code>WeaponFactory</code> class is a factory for weapons.
 * A ship who want a weapon should ask for it from the WeaponFactory.
 * It also creates bullets from bullet model arrived from the newtork.
 */
public class WeaponFactory {

    /** Type of weapons in the game */
    private static final int NUM_OF_PLAYER_WEAPONS = 2;
    public static final int TYPE_LASER_CANNON = 1;
    public static final int TYPE_FIRE_CANNON = 2;

    // Only enemy weapons
    public static final int TYPE_BALL_CANNON = 101;

    // Images for the weapon upgrades
    private static final String laserCannonImage = "laserUp.png";
    private static final String fireCannonImage = "fireUp.png";
    private static final String ballCannonImage = "ballUp.png";

    /**
     * Returns a random weapon type.
     */
    public static int getRandomWeaponType() {
        return 1 + (int)(Math.random() * NUM_OF_PLAYER_WEAPONS);
    }

    public static String getWeaponImageByType(int type) {
        switch (type) {
            case TYPE_LASER_CANNON:
                return laserCannonImage;
            case TYPE_FIRE_CANNON:
                return fireCannonImage;
            case TYPE_BALL_CANNON:
                return ballCannonImage;
            default:
                return "";
        }

    }

    /**
     * Returns a <code>Weapon</object>.
     * @param weaponType  Type of the weapon
     * @param weaponLevel Weapon level
     * @param direction   Direction of the weapon
     * @return  A weapon
     */
    public static Weapon getWeapon(int weaponType, int weaponLevel,
            int direction) {

        Weapon weapon = null;

        switch (weaponType) {
            case TYPE_LASER_CANNON:
                weapon = new LaserCannon(direction, weaponLevel);
                break;
            case TYPE_FIRE_CANNON:
                weapon = new FireCannon(direction, weaponLevel);
                break;
            case TYPE_BALL_CANNON:
                weapon = new BallCannon(direction, weaponLevel);
        }

        return weapon;

    }

    /**
     * This method is used to get a <code>Bullet</code> object from a
     * model sent via the network.
     * XXX: The method could be implemented with reflection but it's error
     *  prone and hurts performance
     * @param model Bullet model
     * @param owner Bullet's owning ship
     */
    public static Bullet getBullet(BulletModel model, Ship owner) {
        Bullet bullet = null;
        Class bulletClass = model.bulletClass;
        if (bulletClass == LaserBeam.class) {
            bullet = new LaserBeam(
                    owner, model.x, model.y, model.dx, model.dy);
        }
        else if (bulletClass == FireBullet.class) {
            bullet = new FireBullet(
                    owner, model.x, model.y, model.dx, model.dy);
        }
        else if (bulletClass == BlueLaser.class) {
            bullet = new BlueLaser(
                    owner, model.x, model.y, model.dx, model.dy);
        }
        else if (bulletClass == BallBullet.class) {
            bullet = new BallBullet(
                    owner, model.x, model.y, model.dx, model.dy);
        }
        else if (bulletClass == BigBallBullet.class) {
            bullet = new BigBallBullet(
                    owner, model.x, model.y, model.dx, model.dy);
        }

        return bullet;


        // Sample of doing the same with reflection
        // Assuming all the bullets implement constructor accepting BulletModel
        // and Ship as parameters and the BulletModel contains the bullet class
        /*
        try {
            Class [] params = new Class[]{BulletModel.class, Ship.class};
            Constructor cons = model.bulletClass.getConstructor(params);
            Object [] consParams = new Object[]{model, getNetworkPlayerShip()};
            Bullet bullet = (Bullet)cons.newInstance(consParams);
            addShoot(bullet);
      }
        catch (Exception e) {
            e.printStackTrace();
        }
```

```
         */

    }

}
```

```java
package game.sound;

import javax.sound.sampled.AudioFormat;

/**
 * The <code>CachedSound</code> is used to cache
 * sound file bytes.
 */
public class CachedSound {

    private byte[] samples;
    private AudioFormat audioFormat;

    /**
     * Construct a new <code>CachedSound</code>
     * @param samples Sound samples.
     */
    public CachedSound (byte[] samples, AudioFormat audioFormat) {
        this.samples = samples;
        this.audioFormat = audioFormat;
    }


    /**
     * Returns the sound samples byte array.
     * @return The sound samples byte array.
     */
    public byte[] getSamples() {
        return this.samples;
    }

    /**
     * Returns the sound audio format.
     * @return The sound audio format.
     */
    public AudioFormat getAudioFormat() {
        return this.audioFormat;
    }


}
```

```java
package game.sound;

import javax.sound.sampled.Line;
import javax.sound.sampled.LineEvent;
import javax.sound.sampled.LineListener;

/**
 * This class is used as sound line listener. When a stop
 * line event arrives the line is closed.
 */
public class SimpleLineListener implements LineListener {

    public void update(LineEvent event) {
        if (event.getType() == LineEvent.Type.STOP) {
            Line line = event.getLine();
            line.close();
        }
    }
}
```

```java
package game.sound;

import game.GameConstants;

import java.applet.Applet;
import java.applet.AudioClip;
import java.io.*;

import javax.sound.sampled.*;

/**
 * The <code>SoundFactory</code> class is used to play sounds
 * in the game.
 */
public class SoundFactory {

    private static LineListener lineListener = new SimpleLineListener();

    /**
     * Play the sound file. This method is not blocking.
     * @param fileName  File name with the sound (we serach in the
     * sounds folder).
     */
    public static void playSound(String fileName) {

        try {

            File soundFile = new File(GameConstants.SOUNDS_DIR + fileName);
            AudioInputStream ais = AudioSystem.getAudioInputStream(soundFile);
            AudioFormat audioFormat = ais.getFormat();
            DataLine.Info info = new DataLine.Info(Clip.class, audioFormat);
            // Obtain a line
            Clip clip = (Clip) AudioSystem.getLine(info);
            // Open the line (aquire resources)
            clip.open(ais);
            // Set the line event's listener
            clip.addLineListener(lineListener);
            // Start playing the playback
            clip.start();

        }

        catch (UnsupportedAudioFileException uafe) {
            uafe.printStackTrace();
        }
        catch (LineUnavailableException lue) {
            lue.printStackTrace();
        }
        catch (IOException ioe) {
            ioe.printStackTrace();
        }

    }

    /**
     * Play an audio clip in the "Applet" way (i.e., using the
     * <code>Applet</code> class). This method blocks so we don't
     * use it in the game.
     * @param fileName  Name of the audio clip file.
     */
    public static void playAppletClip(String fileName) {
        try {
          File file = new File(GameConstants.SOUNDS_DIR + fileName);
          AudioClip clip = Applet.newAudioClip(file.toURL());
          clip.play();
        }
        catch (Exception e) {}
    }


}
```

```
package game.util;

import java.awt.Component;

import javax.swing.JOptionPane;

/**
 * The <code>Logger</code> class is a simple logging helper.
 * Most of the game errors and exceptions are sent to the Logger.
 * It also handles the command line parameters.
 */
public class Logger {

    private static boolean debug; // debug mode flag
    private static boolean invulnerable; // player ships vulnerability

    /**
     * Initializes the Logger, check if in debug mode.
     * This method should be called once when the game starts.
     * (can also be used to redirect out streams)
     * @param args  Arguments from the command line
     */
    public static void init(String[] args) {
        for (int i = 0; i < args.length; i++) {
            String param = args[i];
            if (param.equals("debug")) {
                debug = true;
            }
            else if (param.equals("invulnerable")) {
                invulnerable = true;
            }
        }
    }


    /**
     * Print the message to the screen.
     * @param message Message to display.
     */
    public static void screen(String message) {
        System.out.println(message);
    }

    /**
     * Print the error to the screen.
     * @param error Error message to display.
     */
    public static void error(String error) {
        System.err.println(error);
    }

    /**
     * Prints the stack trace of the exception to the screen.
     * @param e Exception to print.
     */
    public static void exception(Exception e) {
        e.printStackTrace();
    }

    /**
```

```
     * Displays a <code>JOptionPane</code> error dialog.
     * @param parent  Parent dialog window.
     * @param message Message to display.
     */
    public static void showErrorDialog(Component parent, String message) {

        JOptionPane.showMessageDialog(parent, message,
                "Error", JOptionPane.ERROR_MESSAGE);

    }

    /**
     * Prints to the screen the prefix and the total used memory if in
     * debug mode.
     * @param prefix  Prefix message
     */
    public static void printMemoryUsage(String prefix) {
        if (isDebug()) {
            Runtime runtime = Runtime.getRuntime();
            long usedMemory = runtime.totalMemory() - runtime.freeMemory();
            screen(prefix + "\t" + usedMemory);
        }
    }

    /**
     * Returns true if the debug flasg is on. The debug flag is
     * taket from the system property.
     * @return  True if in debug mode
     */
    public static boolean isDebug() {
        return debug;
    }

    /**
     * Returns true if the application started with the invulnerable
     * flag on.
     */
    public static boolean isInvulnerable() {
        return invulnerable;
    }

}
```

```java
package game.util;

import game.GameConstants;

import java.awt.*;
import java.io.File;
import java.io.FileInputStream;

import javax.swing.ImageIcon;

/**
 * The <code>ResourceManager</code> class contains some static methods
 * for various resource loading and management.
 */
public class ResourceManager {

    private static Font gameFont;
    private static Cursor gameCursor, invisibleCursor;

    static {
        init();
    }

    /**
     * Initialize and preload resources.
     */
    private static void init() {

        // Create the game font
        try {
          File file = new File(
                  GameConstants.RESOURCES + "/" + GameConstants.GAME_FONT);
          FileInputStream fis = new FileInputStream(file);
          gameFont = Font.createFont(Font.TRUETYPE_FONT, fis);
          fis.close();
        }
        catch (Exception e) {
            Logger.exception(e);
            gameFont = new Font("Serif", Font.PLAIN, 1);
        }

        // Load the game cursors
        invisibleCursor = getCursor("");
        gameCursor = getCursor(GameConstants.IMAGES_DIR +
                GameConstants.GAME_CURSOR);

    }

    /**
     * Returns the game font with the specified font style and size.
     * @param style Style of the font (e.g., Font.BOLD).
     * @param size  Size of the font.
     * @return  Game font with the specified font style and size.
     */
    public static Font getFont(int style, int size) {

        return gameFont.deriveFont(style, size);
    }

    /**
```

```
     * Returns the game font with the PLAIN style and input size.
     * @param size Size of the font.
     * @return  Plain game font with the specified size.
     */
    public static Font getFont(int size) {

        return getFont(Font.PLAIN, size);
    }

    /**
     * Returns a new <code>Cursor</code> object created from the input
     * image name. Expects image of size 32x32 and places the hot spot in
     * the middle of the image.
     * If the requested cursor is invisible or the default one, return the
     * pre-created cursor.
     * @param imageName Name of the image to be loaded as the cursor image
     * @return  A cursor
     */
    public static Cursor getCursor(String imageName) {
        Cursor cursor = null;

        if (imageName.equals("") && invisibleCursor != null) {
            cursor = invisibleCursor;
        }
        else if (imageName.equals(
                GameConstants.GAME_CURSOR) && gameCursor != null) {
            cursor = gameCursor;
        }
        else {
            cursor =
                Toolkit.getDefaultToolkit().createCustomCursor(
                    loadImage(imageName), new Point(16, 16), imageName);
        }

        return cursor;

    }

    /**
     * Loads and returns an image.
     * @param imageName Image name to load
     */
    public static Image loadImage(String imageName) {
        return new ImageIcon(imageName).getImage();
    }
}
```