Università degli Studi di Messina

Dipartimento di Scienze Matematiche, Informatiche, Fisiche e Scienze della Terra
Corso di Laurea Triennale in Informatica

Relazione Programmazione II

A.A. 2022-2023

Monkey Game

Prof. Antonino Galletta

Rando Caterina 468044

INDICE

1 Introduzione

L'idea alla base del progetto è l'implementazione di un gioco sviluppato completamente in Java. Java è uno dei linguaggi di programmazione orientati agli oggetti. Il codice del progetto è disponibile su Github al seguente indirizzo: https://github.com/CRando12/programmazion2

1.1 Monkey Game

Monkey Game è un gioco nel quale il player, rappresentato da una scimmia, può spostarsi verso destra o sinistra con l'obiettivo di raccogliere un determinato numero di frutti, generati in modo randomico, entro un determinato tempo.
Il gioco comprende due modalità di gioco:

- Partita semplice: il player dopo aver avviato il gioco inizia la sua partita;
- Carriera: il player deve autenticarsi per accedere alla propria carriera dove troverà un Marketplace/Shop per acquistare e modificare gli oggetti della partita e tenere traccia dei progressi nel tempo attraverso un Database.

È stata anche implementata una connessione client/server, quando il gioco verrà avviato il server si connetterà e rimarrà in ascolto in attesa di stabilire una connessione con il client.

1.2 Java

Java è un linguaggio di programmazione "ad alto livello" (orientato agli oggetti), definito dalla Sun negli anni '90. Java, mira allo sviluppo di applicazioni sicure, efficienti, robuste, su piattaforme multiple, in reti eterogeneee e distribuite. È un linguaggio interpretato e per la traduzione dei file sorgenti in linguaggio macchina, Java, utilizza un approccio misto: il sorgente scritto in Java, viene compilato (da un compilatore Java) che genera un file scritto in bytecode, il quale viene interpretato (ed eseguito) da un interprete Java (generalmente chiamato *Java Virtual Machine*: JVM o Java Runtime Environment: JRE). Il compilatore e il bytecode sono indipendenti dalla macchina. La JVM dovendo tradurre in linguaggi macchina specifici dipende dalla macchina (S.O. + CPU). Prima di essere eseguito, il codice sorgente viene compilato in bytecode, in modo che possa essere eseguito su diverse architetture, a condizione che la corretta versione del JDK (Java Development Kit) sia installata sulla macchina. JDK include l'interpreter Java, le classi Java e gli strumenti di sviluppo Java. Il JDK consente la scrittura di applicazioni che sono state sviluppate una volta e di effettuare l'esecuzione dovunque su qualunque Java virtual machine. Le applicazioni Java sviluppate con JDK su un sistema possono essere utilizzate su un altro sistema senza modificare o ricompilare il codice.

2 Requisiti

I requisiti richiesti e rispettati per lo sviluppo del progetto sono riportati di sequito:

- Implementazione delle caratteristiche di programmazione II:
  - Incapsulamento e information hiding;
  - Ereditarieà;
  - Polimorfismo.
- Utilizzo delle interfacce;
- Interfacciamento con un database;
- Gestione delle eccezioni;

- Configurazione attraverso un file XML;

2.1 Caratteristiche Programmazione II

2.1.1 Incapsulamento e information hiding

L'incapsulamento fornisce un mezzo per utilizzare la classe ma omette dettagli di come è fatto. Viene anche chiamato *Information Hiding* (nascondimento delle nformazioni), il concetto principale dell'Information Hiding riguarda i metodi delle classi che un programmatore può utilizzare e non interessa conoscere i dettagli implementativi, quello che interessa è quello che fa quell'handle, cosa prende in ingresso e cosa prende in uscite. Questo concetto rientra nella logica del riutilizzo del codice. Quindi, Information Hiding significa nascondere i dettagli implementativi, è il principio per cui si preferisce che alcune informazioni di un oggetto non siano accessibili direttamente dall'esterno del suo contesto. La distinzione tra le informazioni accessibili e le informazioni non accessibili viene stabilita utilizzare le keywords di Java:

```
public class Menu implements ActionListener, Runnable
{
    private JFrame f;
    private JButton newgame;
    private JButton regole;
    private JButton exitgame;
    private JButton carriera;

    public Menu(JFrame f)
    {
        this.f = f;
    }
```

- *Public*: i campi e i metodi che utilizzano questa keyword possono essere richiamati da altre classi tramite la *dottted notation*;
- *Private*: i campi e i metodi che utilizzano questa kewyword non possono essere richiamati da altre classi in quanto privati.

La dotted notation è una modalità di accesso per accedere allo stato dell'oggetto e utilizza una sintassi del tipo *<oggetto>.<campo>*.

2.1.2 Ereditarietà

L'ereditarietà è quel concetto che abbiamo in programmazione a oggetti in cui si eredita qualcosa. Si ha una classe generale che via via si va a specializzare sempre di più. Viene definita una classe base (padre) che viene chiamata *superclasse* e/o *classe generica*, si definiscono le classi derivate dalla superclasse che vengono chiamate *sottoclassi* e/o *classi specializzate* (figli). Tutto quello che è defnitio all'interno della superclasse viene ereditato dalle sottoclassi trannne i metodi e le variabili che sono state dichiarate con la keyword private. Una classe di livello inferiore eredita tutto quello che c'è nella classe di livello superiore, tutto quello che viene ereditato non c'è bisogno che venga ripetuto. Ogni classe può aggiungere delle proprie caratteristiche a quelle già esistenti.

```
public class Rectangle extends Shape
{
    private int r = 255, g = 255, b = 255; //Default bianco
    private int opacity = 0;

    public Rectangle(){}

    @Override
    public void paintComponent(Graphics g)
    {
        Dimension dim = getSize();
        g.setColor(new Color(r: this.r, g: this.g, b: this.b, a: this.opacity));
        g.fillRect(x: 0, y: 0, width: dim.width, height:dim.height);
    }
```

2.1.3 Polimorfismo

Il polimorfismo ci consente di avere dei cambiamenti nella definizione del metodo delle classi derivate e avere questi cambiamenti dei metodi scritti. Al polimorfismo vengono associati due concetti:

- *Overload:* si riferisce a due metodi di una stessa classe che hanno lo stesso nome ma *signature* (firme) differenti;
- *Override:* si riferisce a una classe derivata che modifica l'implementazione di un metodo della superclasse però mantiene la stessa firma e crea in ogni caso un'entità distinta definita nella classe derivata. In questo modo le sottoclassi possono personalizzare le caratteristiche ereditate dalla superclasse.

La signature (firma) di un metodo è l'insieme delle informazioni che identificano in modo univoco quel metodo, compresa la lista dei parametri e l'ordine in cui sono dichiarati.

```
@Override
public void actionPerformed(ActionEvent e)
{
    //Prendo il bottone che ha scatenato l'evento
    Object source = e.getSource();
    //A seconda del bottone eseguo codice differente
    if(source.equals(obj:hocapito))
    {
        Sounds click = new Sounds(nameSound:"click.wav");
        click.start();
        Menu m = new Menu(f);
        m.inizia();
    }
}
```

2.2 Database

È stata definita una classe per la gestione della connessione al database MySQL denominata '*DB.java*'. La classe restituisce un oggetto di tipo *Connection* che rappresenta la connessione al database. I dati per la connessione al database verranno prelevati dal file denominato *config.xml*. Alla fine verrà restituita la connessione dall'oggetto *conn* tramite il metodo *getConn()* e sarà un oggetto di tipo Connection.

```
public Db()
{
    conn = null;

    try {
        File configFile = new File(pathname: "src/monkey/config.xml");

        DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
        DocumentBuilder dBuilder = null;
        try {
            dBuilder = dbFactory.newDocumentBuilder();
        } catch (ParserConfigurationException ex) {
            Logger.getLogger(name: Db.class.getName()).log(level:Level.SEVERE, msg: null, thrown: ex);
        }

        Document doc = null;
        try {
            doc = dBuilder.parse(f: configFile);
        } catch (SAXException ex) {
            Logger.getLogger(name: Db.class.getName()).log(level:Level.SEVERE, msg: null, thrown: ex);
        } catch (IOException ex) {
            Logger.getLogger(name: Db.class.getName()).log(level:Level.SEVERE, msg: null, thrown: ex);
        }

        doc.getDocumentElement().normalize();

        NodeList nodeList = doc.getElementsByTagName(tagname: "config");

        for (int temp = 0; temp < nodeList.getLength(); temp++)
        {
            Node node = nodeList.item(index:temp);

            if (node.getNodeType() == Node.ELEMENT_NODE) {
                Element element = (Element) node;

                HOST = element.getElementsByTagName(name: "host").item(index:0).getTextContent();
                USERNAME = element.getElementsByTagName(name: "username").item(index:0).getTextContent();
                PASSWORD = element.getElementsByTagName(name: "password").item(index:0).getTextContent();
                DBNAME = element.getElementsByTagName(name: "dbname").item(index:0).getTextContent();
```

```
        if (node.getNodeType() == Node.ELEMENT_NODE) {
            Element element = (Element) node;

            HOST = element.getElementsByTagName( name: "host").item( index: 0).getTextContent();
            USERNAME = element.getElementsByTagName( name: "username").item( index: 0).getTextContent();
            PASSWORD = element.getElementsByTagName( name: "password").item( index: 0).getTextContent();
            DBNAME = element.getElementsByTagName( name: "dbname").item( index: 0).getTextContent();

            // Usa i valori estratti come desideri
            System.out.println("Host: " + HOST);
            System.out.println("Username: " + USERNAME);
            System.out.println("Password: " + PASSWORD);
            System.out.println("DB Name: " + DBNAME);
        }
    }

    CONN_STRING = "jdbc:mysql://" + HOST + "/" + DBNAME;
    conn = DriverManager.getConnection( url: CONN_STRING, user: USERNAME, password: PASSWORD);

} catch (SQLException e) {
    System.err.println( x: "Connessione al database fallita.");
    System.err.println( x: "Controlla di aver aperto XAMPP o il server MySQL!");
    System.out.println( x: e);
}
```

```
public Connection GetConn(){
    return conn;
}
```

2.3 Gestione delle eccezioni

L'eccezione è un oggetto che permette di gestire in modo corretto degli eventi anomali. Per controllare un'eccezione si utilizza la clausola *try* seguita da uno o più clausole *catch*. Le clausole rappresentano:

- *Try:* blocco di istruzioni che possono generare situazioni di errore, se l'esecuzione del blocco non incontra errori, passa all'istruzione successiva al blocco *catch;*
- *Catch:* nel caso in cui, nel blocco *try* vi sono degli errori, il blocco *catch* prova a gestire nel modo corretto l'eccezione.

Nel caso riportato di fianco, nel momento in cui il gioco verrà avviato si avvierà un musica che verrà cercata nel path indicato, nel caso in cui il file non viene trovato verrà generata un'eccezione e viene lanciato un messaggio di errore.

```
public Sounds(String nameSound)
{
    try
    {
        // Ricerco nella cartella /src/sound/ le varie track .wav
        File fx = new File("./src/sounds/" + nameSound);
        AudioInputStream audioIn = AudioSystem.getAudioInputStream( url: fx.toURI().toURL());
        clip = AudioSystem.getClip();
        clip.open( stream: audioIn);
    }
    catch (Exception e)
    {
        System.out.println( x: "Errore, suono non riproducibile, motivo:");
        System.err.println( x: e.getMessage());
    }
}
```

2.4 XML

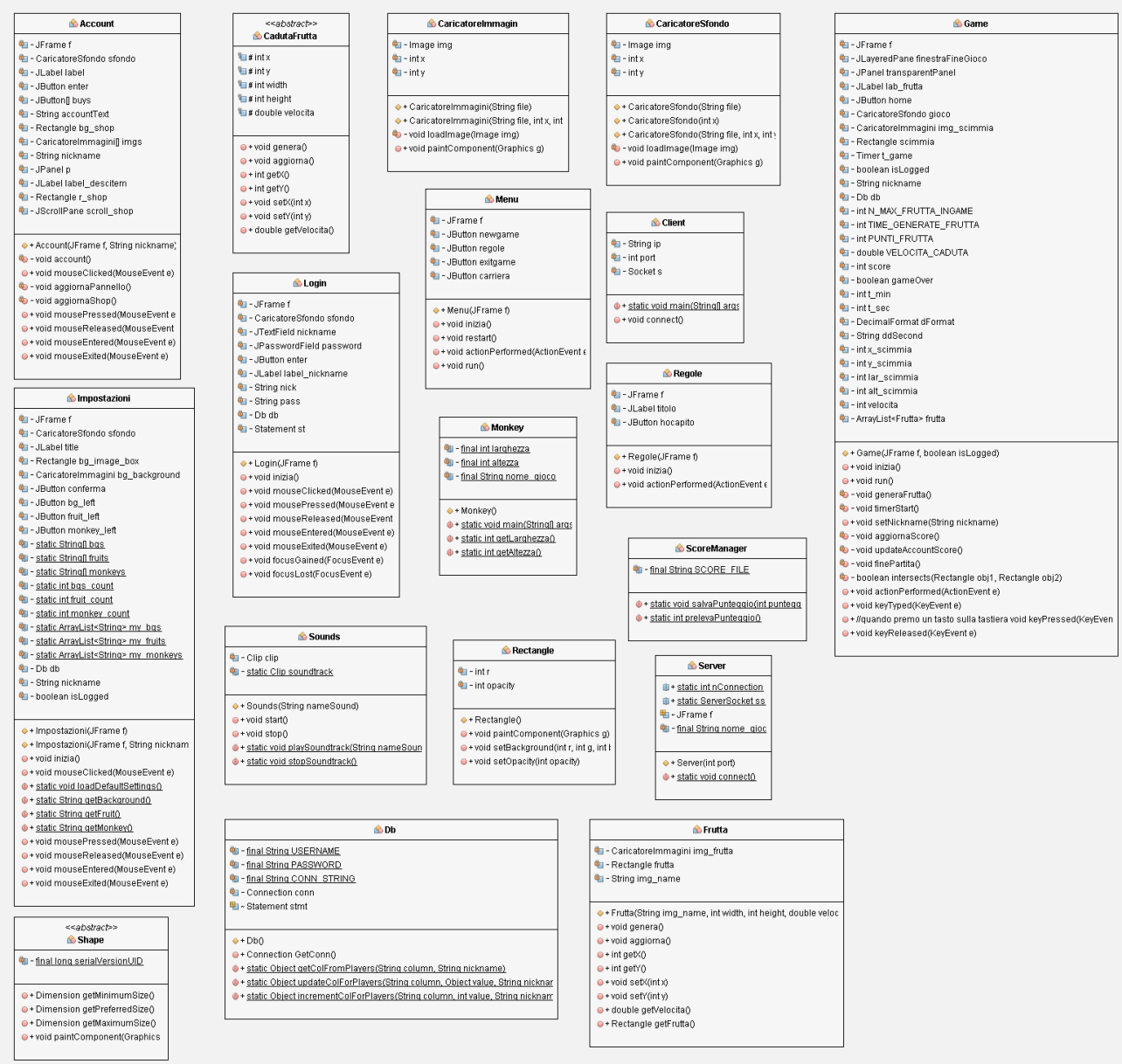
XML è un linguaggio che permette la scrittura di file di testo contenenti dati per la comunicazione tra i componenti software. Il file denominato *conf.xml* contiene al suo interno i dati necessari per stabile la connessione al database. In particolare, troviamo: host, pota, nome del database, username e password.

```
<?xml version="1.0" encoding="UTF-8"?>

<config>
    <db>
        <host>localhost</host>
        <port>3306</port>
        <name>monkey</name>
        <username>root</username>
        <password></password>
    </db>
</config>
```

3 Diagramma delle classi

Il diagramma delle classi è un diagramma strutturato secondo lo standard UML (Unified Modeling Language) viene utilizzato per descrivere una classe. Viene inserito il nome della classe, le variabili e i metodi. Ad ogni variabile e metodo (con relativa signature) vengono definite tramite un simbolo (+ pubblica, - privata) le regole di visibilità. Le variabili locali non vengono inserite nel diagramma UML. Vengono anche indicate le relazioni di ereditarietà e le implementazioni delle interfacce.



4 Funzionalità

Il gioco è stato realizzato per permettere al player di poter giocare e conservare i progressi effettuati ad ogni partita e per farlo è stato aggiunto:

- Login: permette all'utente di accedere alla propria carriera inserendo nome utente e la password;
- Impostazioni: permette all'utente di visualizzare lo shop e acquistare/ modificare gli oggetti di gioco.

4.1 Login

La classe ‘Login.java’ implementa le classi ‘MouseListener’ e la classe ‘FocusListener’. Viene impostata la schermata iniziale tramite *JFrame f* e imposto lo sfondo utilizzando la classe CaricatoreSfondo. Vengono create delle label per il nickname e la password, e delle aree di testo per permettere all’utente di inserire i valori, il bottone per accedere all’area dell’utente.

```
public class Login implements MouseListener, FocusListener {

    private JFrame f;
    private CaricatoreSfondo sfondo;
    private JTextField nickname;
    private JPasswordField password;
    private JButton enter, back, newaccount;
    private JLabel label_nickname, label_password, error;

    private String nick = "";
    private String pass = "";

    // Variabili del database
    private Db db;
    private Statement st;
```

All’interno del metodo *inizia()* vengono create delle label per il nickname e la password, e delle aree di testo per permettere all’utente di inserire i valori, il bottone per accedere all’area dell’utente. Nel metodo *mouseClicked(MouseEvent e)* della classe MouseEvent di Event. Gestisce gli eventi causati da click. Viene effettuato un controllo sul testo della password, viene eseguita la query e se la password scritta dall’utente combacia con quella registrata nel database si passa al controllo del nickname, se anche questo va a buon fine viene verificato si procede con l’accesso all’area carriera, altrimenti ci sarà un messaggio di errore.

```
public void inizia()
{
    f.getContentPane().removeAll();
    f.repaint();

    sfondo = new CaricatoreSfondo( file: "sfondo1.jpg");
    f.getContentPane().add( comp: sfondo);

    //Impostiamo un layout libero
    sfondo.setLayout( mgr: null);

    f.pack();
    f.setVisible( b: true);

    new GetFont( fontsource: "Waverly.ttf", dimension: 20f);

    label_nickname = new JLabel( text: "Nickname: ");
    label_nickname.setBounds( x: 515, y: 170, width: 220, height: 40);
    label_nickname.setForeground( fg: Color.WHITE);
    label_nickname.setFont( font: GetFont.GetThisFont().deriveFont( style: Font.BOLD));

    label_password = new JLabel( text: "Password: ");
    label_password.setBounds( x: 515, y: 280, width: 220, height: 40);
    label_password.setForeground( fg: Color.WHITE);
    label_password.setFont( font: GetFont.GetThisFont().deriveFont( style: Font.BOLD));

    nickname = new JTextField();
    nickname.setBounds( x: 515, y: 230, width: 220, height: 40);
    nickname.setHorizontalAlignment( alignment: JTextField.CENTER);
    nickname.setEditable( b: true);

    password = new JPasswordField();
    password.setBounds( x: 515, y: 340, width: 220, height: 40);
    password.setHorizontalAlignment( alignment: JTextField.CENTER);
    password.setEditable( b: true);

    enter = new JButton( text: "Entra");
    enter.setBounds( x: 515, y: 420, width: 220, height: 40);
    enter.addMouseListener( l: this);
```

```
@Override
public void mouseClicked(MouseEvent e)
{
    enter.setEnabled(b: false);
    error.setText(text: "Connessione ...");
    error.setForeground(fg: Color.green);
    error.setVisible(aFlag: true);

    if(e.getSource() == enter)
    {
        char[] keyword = password.getPassword();
        pass = String.valueOf(data: keyword);
        nick = nickname.getText();

        ResultSet rs;

        String checkForUser = "SELECT * FROM players WHERE nickname = '" + nick + "'";
        String checkUserAndPass = "SELECT * FROM players WHERE nickname = '" + nick + "' AND password = '" + pass + "'";

        if(!"".equals(anObject: nick) && !"".equals(anObject: pass))
        {
            try {
                st = (Statement) db.GetConn().createStatement();
                rs = st.executeQuery(sql: checkForUser);

                if(rs.next())
                {
                    try {
                        st = (Statement) db.GetConn().createStatement();
                        rs = st.executeQuery(sql: checkUserAndPass);

                        if(rs.next())
                        {
                            // Se nickname e password sono corretti
                            // Entro nell'account del player
                            Account a = new Account(f, nickname: nick);
```

Se il nickname non è presente sui sistemi del database viene segnalato

```
                        } catch (SQLException ex){
                            Logger.getLogger(name: Login.class.getName()).log(level: Level.SEVERE, msg: null, thrown: ex);
                        }
                    }
                    else
                    {
                        error.setText(text: "<html>Errore: il nome utente non esiste.\n<br>Creare un nuovo account?</html>");
                        error.setForeground(fg: Color.red);
                        error.setVisible(aFlag: true);
                        enter.setEnabled(b: true);
                        newaccount.setVisible(aFlag: true);
                    }

                    st.close();

                }
                catch (SQLException ex)
                {
                    Logger.getLogger(name: Login.class.getName()).log(level: Level.SEVERE, msg: null, thrown: ex);
                }
            }
            else
            {
                error.setText(text: "Attenzione: compila tutti i campi");
                error.setForeground(fg: Color.red);
                error.setVisible(aFlag: true);
                enter.setEnabled(b: true);
                newaccount.setVisible(aFlag: false);
            }
        }
        else if(e.getSource() == newaccount)
        {
            String newUserData = "INSERT INTO players (nickname, password, level, score, played, credits) VALUES ('"+nick+"', '"+pass+"', '1', '0', '0', '0')";

            try {
                db.GetConn().createStatement().executeUpdate(sql: newUserData);
                // Entro nell'account del player
                Account a = new Account(f, nickname: nick);
            }
            catch (SQLException ex)
            {
                Logger.getLogger(name: Login.class.getName()).log(level: Level.SEVERE, msg: null, thrown: ex);
            }
        }
        else if(e.getSource() == back)
        {
            Sounds.stopSoundtrack();
            Menu m = new Menu(f);
            m.restart();
        }
    }
```

l'errore e verrà data la possibilità di registrarsi andando a compilare tutti i campi.

4.2 Impostazioni

Nella classe *Impostazioni.java* viene implementato MouseListener.
Il costruttore *Impostazioni (JFrame f, String nickname)* imposta la schermata di gioco ed effettua la connesssione al database quan do *isLogged=true*.

```
public class Impostazioni implements MouseListener {

    private JFrame f;
    private CaricatoreSfondo sfondo;
    private JLabel title, lab_score;
    private Rectangle bg_image_box, fruit_image_box, monkey_image_box;
    private CaricatoreImmagini bg_background, fruit_background, monkey_background;
    private JButton conferma;
    private JButton bg_left, bg_right;
    private JButton fruit_left, fruit_right;
    private JButton monkey_left, monkey_right;

    // Vettore di background
    private static String[] bgs = {"sfondo0.jpg", "sfondo1.jpg", "sfondo2.jpg", "sfondo3.jpg", "sfondo4.jpg"};
    private static String[] fruits = {"banana.png", "fragola.png", "mela.png", "pera.png"};
    private static String[] monkeys = {"scimmia0.png", "scimmia1.png", "scimmia2.png"};
    private static int bgs_count = 0;
    private static int fruit_count = 0;
    private static int monkey_count = 0;

    // Nuove liste
    private static ArrayList<String> my_bgs = new ArrayList();
    private static ArrayList<String> my_fruits = new ArrayList();
    private static ArrayList<String> my_monkeys = new ArrayList();

    // Connessione al db
    private Db db;

    // User-data
    private String nickname;
    private boolean isLogged = false;
```

```
public Impostazioni(JFrame f, String nickname)
{
    this.f = f;
    this.nickname = nickname;
    this.isLogged = true;

    if(isLogged)
    {
        // Apro la connessione al database
        db = new Db();
    }
}
```

Vengono caricati gli oggetti acquistati dall'utente tramite una query verificando quali sono gli oggetti acquistati.

```
// Caricamento degli acquisti dell'utente
try {
    String playerItemsQuery = "SELECT * FROM items";
    Statement stmt1 = (Statement) db.GetConn().createStatement();
    ResultSet items = stmt1.executeQuery(sql:playerItemsQuery);

    while(items.next())
    {
        int id_item = items.getInt(columnLabel: "id");

        String checkPlayerItem = "SELECT * FROM player_item WHERE id = '" + id_item + "' AND nickname = '" + nickname + "'";
        Statement stmt2 = (Statement) db.GetConn().createStatement();
        ResultSet playerItem = stmt2.executeQuery(sql:checkPlayerItem);

        // Se l'utente possiede l'oggetto, aggiungilo dall'array
        if(playerItem.next())
        {
            String item_type = items.getString(columnLabel: "type");
            String item_url = items.getString(columnLabel: "img_url");

            switch(item_type)
            {
                case "fruit":
                    my_fruits.add(e: item_url);
                    break;
                case "background":
                    my_bgs.add(e: item_url);
                    break;
                case "monkey":
                    my_monkeys.add(e: item_url);
                    break;
            }
        }
    }
} catch (SQLException ex){
    Logger.getLogger(name: Login.class.getName()).log(level: Level.SEVERE, msg:null, thrown: ex);
}
```

Nel metodo *mouseClicked (MouseEvent e)* vengono salvate frecce (destra e sinistra) che permettono all'utente di scorrere lo sfondo e i frutti acquistati dall'utente per poter andare a modificare lo scenario di gioco.

Il metodo *loadDefaultSetting()* è un metodo statico che imposta gli elementi di default nella posizione 0 degli Array.

```
public static void loadDefaultSettings()
{
    bgs_count = 0;
    fruit_count = 0;
    monkey_count = 0;

    // Copio gli elementi nelle liste nuove
    my_bgs.add(bgs[0]);
    my_fruits.add(fruits[0]);
    my_monkeys.add(monkeys[0]);
}
```

```
@Override
public void mouseClicked(MouseEvent e)
{
    // Frecce per il background
    if(e.getSource() == bg_left)
    {
        bgs_count--;

        if(bgs_count < 0)
            bgs_count = (my_bgs.size()-1);

        bg_image_box.remove(comp: bg_background);
        bg_background = new CaricatoreImmagini(file: my_bgs.get(index: bgs_count), x: 300, y: 170);
        bg_background.setBounds(x: 10, y: 10, width: 300, height: 170);
        bg_background.setOpaque(isOpaque: false);
        bg_image_box.add(comp: bg_background);
    }
    else if(e.getSource() == bg_right)
    {
        bgs_count++;

        if(bgs_count > (my_bgs.size()-1))
            bgs_count = 0;

        bg_image_box.remove(comp: bg_background);
        bg_background = new CaricatoreImmagini(file: my_bgs.get(index: bgs_count), x: 300, y: 170);
        bg_background.setBounds(x: 10, y: 10, width: 300, height: 170);
        bg_background.setOpaque(isOpaque: false);
        bg_image_box.add(comp: bg_background);
    }
    else if(e.getSource() == fruit_left)
    {
        fruit_count--;

        if(fruit_count < 0)
            fruit_count = (my_fruits.size()-1);

        fruit_image_box.remove(comp: fruit_background);
        fruit_background = new CaricatoreImmagini(file: my_fruits.get(index: fruit_count), x: 100, y: 100);
        fruit_background.setBounds(x: 10, y: 10, width: 100, height: 100);
        fruit_background.setOpaque(isOpaque: false);
        fruit_image_box.add(comp: fruit_background);
    }
    else if(e.getSource() == fruit_right)
    {
        fruit_count++;

        if(fruit_count > (my_fruits.size()-1))
            fruit_count = 0;
```