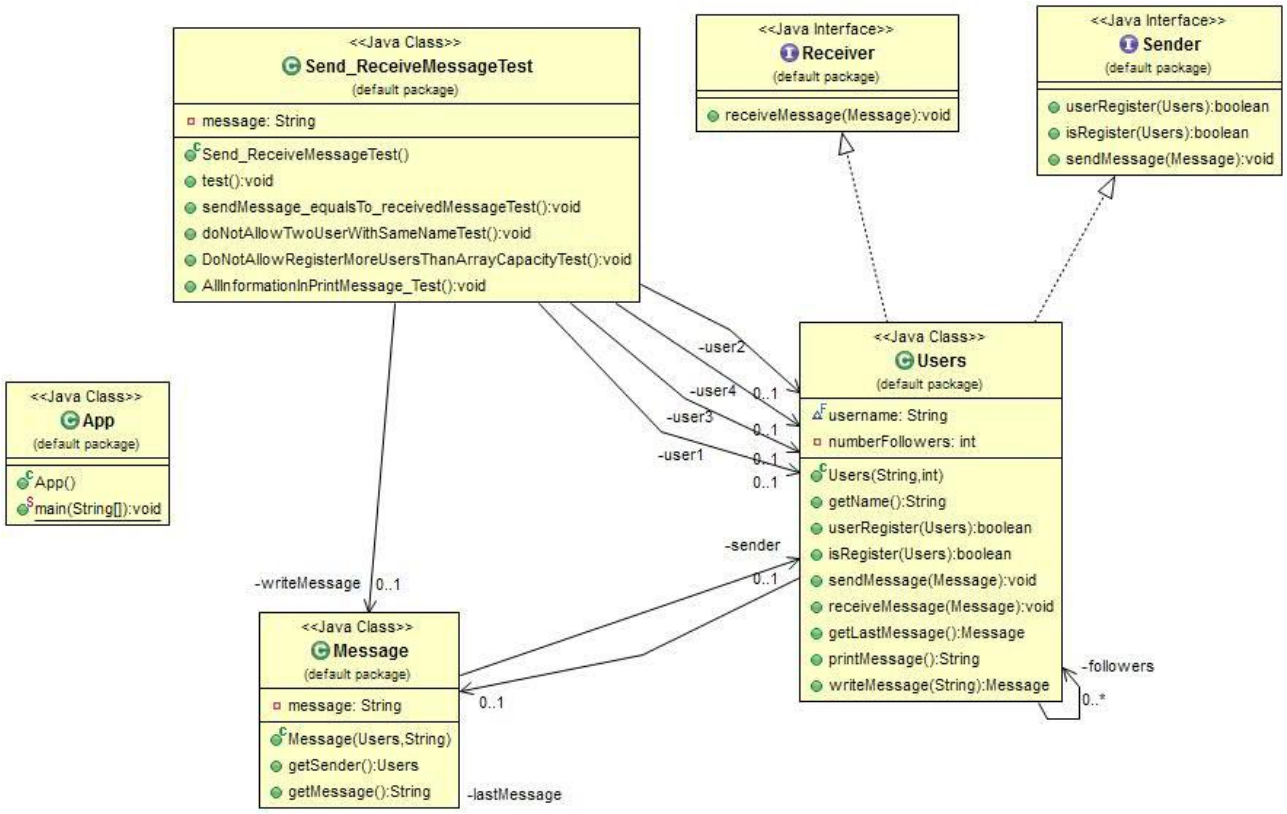
Projecto 12 - Herança e Polimorfismo III (04/11/2014)

1. .




```
/**
 * Receiver - any user that wishes to be notified when the state of another user changes,
 * in this case, want to be notified when the other user sends messages
 * @author Filipa
 */
public interface Receiver {
	/**
	 * Allows the receiver to receive the message
	 * @param msg
	 */
	void receiveMessage(Message msg);
}
```

```
/**
 * Sender -any user whose state may be of interest, and in whom another user may register
 * an interest
 * @author Filipa
 */
public interface Sender {
	/**
	 * This method stores the users
	 * @param User
	 */
	public boolean userRegister(Users user) ;

	/**
	 * Verify if the user is already registered
	 * @param user user's name
	 * @return true if the user is already registered, false otherwise
	 */
	public boolean isRegister(Users user);

	/**
```

```
         * The sender send the message to the receivers
         * @param msg
         * @return
         */
        public void sendMessage(Message msg);
}
```

```
public class Users implements Sender, Receiver{
        private Users[] followers;//Constructor with initial capacity
        final String username;          //nome do user
        private Message lastMessage;      //última mensagem recebida
        private int numberFollowers= 0;       //nº inicial de users registados

        /**
         * The constructor has the following parameter
         * @param name user's name
         * @param numFollowers
         */
        public Users(String name, int maxFollowers) {
                username = name;
                followers = new Users[maxFollowers];
                                                        // lista dos seguidores de determinado user
        }

        /**
         * Return the user's name.
         */
        public String getName() {
                return username;
        }


/**
 * This method stores the users as followers of a specific user
 * Adds an user/receiver to the set of receiver for this user's messages, provided that it is not
 * the same as some observer already in the set.
 * @param User
 */
        public boolean userRegister(Users user)
        {
                if (isRegister(user)){  // se o utilizador já está registado avisa na consola
                        System.out.println("User already registered");
                        return false;
                }

                if(numberFollowers < followers.length)
        // registar o user, mas apenas se o nº de users for inferior à dimensão do contentor sender
                {
                   followers[numberFollowers]= user;
                   numberFollowers++;
                   return true;
                       // só retorna true caso tenha sido corretamente adicionado ao contentor sender
                }
                else
                   System.out.println("This sender is full.");
                return false;
        }

        /**
         * Verify if the user is already registered
         * @param user user's name
         * @return true if the user is already registered, false otherwise
         */
        public boolean isRegister(Users user){
                for(int idx = 0; idx<numberFollowers; idx++)
                                                // verificar se o utilizador já está registado
                   if (followers[idx].username.equals(user.username))
```

```
                    return true;
                return false;
        }

        /**
         * The sender send the message to the receivers
         * @param receiver
         * @param msg
         * @return
         * @return
         */
        @Override
        public void sendMessage(Message msg) {
                        // envia a mesma mensagem para todos os user registados no array
                for(int idx = 0; idx<numberFollowers; idx++)
                        this.followers[idx].receiveMessage(msg);
                //O envio da mensagem pressupõe a receção da mesma pelos followers user
        }

        /**
         * Allow the User to receive a message.
         */
        @Override
        public void receiveMessage(Message msg){
                lastMessage =  msg;     // Guarda a mensagem recebida como LastMessage
// não a guardei num array porque no enunciado refere que o utilizador apenas guarda 1 (a última)
        }

        /**
         * Allow the user to get the last message
         * @return
         */
        public Message getLastMessage(){
                return lastMessage;        //Permite que o utilizador veja a última mensagem
        }

/**
 * Print the last Message received with the sender and receiver information (username) and the text
 * message.
 * @return
 */
//Alterei ligeiramente o print da mensagem para que se percebesse quem recebeu a mensagem (testes e
aplicação)
        public String printMessage (){
           String viewMessage = "To " + this.username + " From: "+
lastMessage.getSender().username + " " + lastMessage.getMessage();
           System.out.println(viewMessage);
           return viewMessage;
        }

        /**
         * This Method allows the user to create a message.
         * @return writemsg
         */
//transforma a String mensagem inserida pelo utilizador na writemsg do Tipo Message, que além do
texto da mensagem, tem também a informação sobre o user que a enviou.
        public Message writeMessage(String msg) {
                Message writemsg = new Message(this, msg);
                return writemsg;
        }
}
```

```java
/**
 * Defines the class Message whose objects represent text messages send between users and
 * the user writer.
 * @author FilipaG
 */
public class Message {
	private Users sender;
	private String message;

	public Message (Users sender, String msg){
		this.sender = sender;
		message = msg;
	}

	/**
	 * Returns the name of the user who sends the message
	 * @return sender
	 */
	public Users getSender() {	// obtém o emissor da mensagem
		return sender;
	}

	/**
	 * Returns the text of the message
	 * @return message
	 */
	public String getMessage(){	// Obtem a o texto da Mensagem
		return message;
	}
}
```

```java
// Para enviar as mensagens é necessário em primeiro lugar definir um emissor.
// Um emissor pode registar vários utilizadores, até um máximo estabelecido previamente.

	import java.util.Scanner;

	public class App {
		public static void main(String[] args) {

			Scanner scanner = new Scanner(System.in);

			// afetação das variáveis a objectos do tipo Users
			Users user1 = new Users("user1", 10);
			Users user2 = new Users("user2", 10);
			Users user3 = new Users("user3", 10);
			Users user4 = new Users("user4", 10);
			Users user5 = new Users("user5", 10);

			// Registo de users
			user1.userRegister(user1);
			user1.userRegister(user2);
			user1.userRegister(user3);
			user2.userRegister(user1);
			user2.userRegister(user2);
			user2.userRegister(user3);

			System.out.println("Message: ");	//escrita da mensagem
			String msg = scanner.nextLine();	//inserção da linha de instruções
```

```
                    // criação de uma mensagem com a informação da mensagem inserida
                    Message writemsg = new Message(user1, msg);
                    user1.sendMessage(writemsg);
                    user3.getLastMessage();
                    user3.printMessage();
                    user2.getLastMessage();
                    user2.printMessage();
                    user1.getLastMessage();
                    user1.printMessage();

                    System.out.println("Message: ");
                    String msg2 = scanner.nextLine();   //inserção da linha de instruções
                    Message writemsg2 = user2.writeMessage(msg2);
                    user2.sendMessage(writemsg2);
                          // criação de uma mensagem com a informação da mensagem inserida
                    user3.getLastMessage();
                    user3.printMessage();
                    user2.getLastMessage();
                    user2.printMessage();
                    user1.getLastMessage();
                    user1.printMessage();

                    scanner.close();
              }
}
```

```
import org.junit.Assert;
import org.junit.Before;
import org.junit.Test;
/**
 * User:
 * - is identity by is name.
 * - it can receive an send messages.
 * - only saves the last message.
 *
 * Message:
 * - its identity by the user writer and its message.
 */
public class Send_ReceiveMessageTest
{
       private Users user1;
       private Users user2;
       private Users user3;
       private Users user4;
       private String message;
       private Message writeMessage;

       @Before
       public void test()
       {
              //Arrange
              user1 = new Users("user1",5);
              user2 = new Users("user2", 5);
              user3 = new Users("user3", 5);
              user4 = new Users("user4", 5);

              user1.userRegister(user1);
              user1.userRegister(user2);
              user1.userRegister(user3);
              user1.userRegister(user4);
```

```
                message = "olá a todos!";
                writeMessage = user1.writeMessage(message);
        }

        /**
         * This test verify if the received message and the send message are equal.
         */
        //verifica se a mensagem enviada é igual à mensagem recebida
        @Test
        public void sendMessage_equalsTo_receivedMessageTest()
        {
                //Act
                user1.sendMessage(writeMessage);
                //Assert
                Assert.assertTrue( user2.getLastMessage().equals(writeMessage) );
        }

        /**
         * This test verify that is not possible register multiples users with the same
name
         */
        @Test
        public void doNotAllowTwoUserWithSameNameTest()
        {
                //Arrange
                Users uNew1 = new Users("user1", 5);
                //Assert
                Assert.assertFalse( user1.userRegister(uNew1) );
                                                        // porque é um objeto já registado
        }

/**
 * This test verify that is not possible to register more user than the arrays capacity
 */
        @Test
        public void DoNotAllowRegisterMoreUsersThanArrayCapacityTest()
        {
                //Arrange
                Users user5 = new Users("user5",5);
                Users user6 = new Users("user6", 5);
                //Act
                user1.userRegister(user5);
                user1.userRegister(user6);
                //Assert
                Assert.assertFalse( user1.userRegister(user6) );
                                        // porque é o contentor de followers já está cheio
        }

        /**
         * This test verify that the printed message have the information expected.
         */
        @Test
        public void AllInformationInPrintMessage_Test()
        {
                //Arrange
                String msg = "To user2 From: " + writeMessage.getSender().username + " " +
writeMessage.getMessage();
                //Act
                user1.sendMessage(writeMessage);
                            // criação de uma mensagem com a informação da mensagem inserida
                user2.getLastMessage();
                String msgActual = user2.printMessage();

                Assert.assertEquals(msg,msgActual);
        }
}
```