Skapad av:Armin Vojnikovic

För godkänt-(G) på projektet krävs del: 1 och 2 av uppgiften
För väl-godkänt(VG) krävs del 1,2 och 3 av uppgiften

1. – Getting cursor keys from the console

1. Create a new project and import the **com.googlecode.lanterna** library (V2.1.9) as we did in the "Chessboard" task.
2. Start with this code

```
package com.company;

import com.googlecode.lanterna.*;
import com.googlecode.lanterna.input.Key;
import com.googlecode.lanterna.terminal.Terminal;

import java.nio.charset.Charset;

public class Main {

    public static void main(String[] args) throws InterruptedException {

        Terminal terminal = TerminalFacade.createTerminal(System.in,
                            System.out, Charset.forName("UTF8"));
        terminal.enterPrivateMode();

        while(true){

            //Wait for a key to be pressed
            Key key;
            do{
                Thread.sleep(5);
                key =terminal.readInput();
            }
            while(key == null);

            System.out.println(key.getCharacter()+ " " + key.getKind());
        }
    }
}
```

3. Run the application and you see that each time you press a key in the window, the corresponding key details are shown in the IntelliJ console.

   We are going to use this information in the following steps.

4. But first we create a new class named **Player** that will contain two **public** integer **fields**, named **x** and **y**. These two fields can be set from the **constructor**.

5. In the main application create a new Player instance named player at position (10,10);
6. Now create a new **private static** method in the main class named **MovePlayer** that takes the player instance as input parameter and returns void. It also need to take one **Terminal** instance.
7. Move the "Wait for a key press" logic to this new method so that the while loop in the main method is empty.
8. Add a switch statement in this method that checks if the cursor keys (left,up,right,down) is pressed and if so updates the (x,y) coordinates of the player accordingly. The Switch statement can be written like : **`switch(key.getKind())`**
9. Add logic to prevent that the player is moved if the (X or Y) position is <0 or >20

10. Add the while loop in the main method, the following:

```
while(true){

    UpdateScreen(player, terminal);
    MovePlayer(player, terminal);
}
```

11. Let's implement the **UpdateScreen** method that:
    a. Clears the terminal window by calling the **ClearScreen** method on the terminal instance.
    b. Plots the position of the player on the screen using this code:

```
terminal.moveCursor(x,y);
terminal.putCharacter('O');


terminal.moveCursor(0,0);
```

(Setting the cursor to 0,0 is a way to always having it in the same position afterwards)

12. Now run the application and you should now be able to move the **O** on the screen using the cursor keys! Try to make sure it does not crash when it hits the edge of the screen.

2 – Monsters!

1. The simple game in the previous exercise is not that exciting without some creepy monsters chasing you. Let's add that!
2. Create a new **Enemy** class that contains two public **x,y** field and a constructor to set them both. Just like the player field.

3. At the top of the main method create a new Enemy instance at position (5,5).
4. In the main while loop add this code:

```
GameLogic(player, enemy);
```

5. Implement the static method **GameLogic** and add logic to move the enemy towards the position of the player. Make the enemy to move one step closer to the player each time the method is called. How you implement this logic is up to you.
6. Modify the **UpdateScreen** method so you also can pass in the Enemy instance and draw it on the screen, perhaps using an **X** character to represent the enemy.

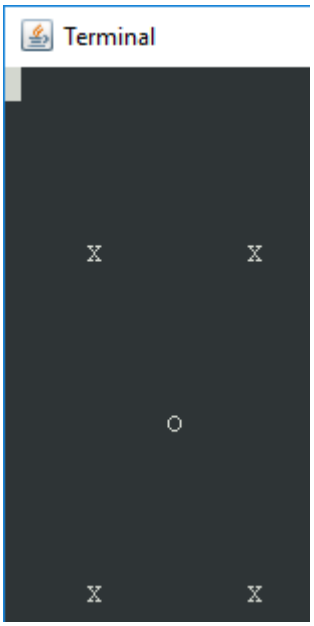
7. Add logic to the **GameLogic** class so that it returns a boolean to represent if the game is over or not. The game is over if the enemy is at the same position as the player.
8. Display **Game Over** on the screen when it ends. To print out a string you need to make your own **PrintString** method. The print screen method needs to take a string and then loop over each character and print it out. Use a method signature like:

```
private static void PrintText(int x, int y, String message,
                              Terminal terminal)
```

Make sure the main while loop exists when the game is over and that the game over message is displayed.

3 – Improving the monster game!

1. In previous exercise we did a simple game where the player is chased by a monster. This time we are going to improve the game by adding support for any number of monsters in the game!
2. Create a new project and copy the code from exercise 13.3
3. Update the **GameLogic** method so that it takes a list of enemies instead of a single enemy and update the code so that it updates all the enemies in the list when the method is called.
4. Update the **UpdateScreen** method so that it takes a list of enemies instead of a single enemy and update the code so that it prints out all the enemies on the screen.
5. In the main method, create a list of Enemies at different positions and pass the list to the methods in the main while loop.
6. Now run the game and see if you can escape the monsters!



7. As you notice the monsters are very fast and let's try to slow them down a bit. To do this we can **replace** the **integer** data type of the enemy **x,y position** with a **float** instead. Do this change.

8. Modify the code so that the monsters only move like 0.5 or 0.7 each time they move. If you need to convert a float to an integer, you can always cast it using this structure:

```
float floatnum = 3.14f;
int intum = (int)floatnum;
```

If you have time, you can enhance this game with features like:

- Only one enemy per cell on the screen
- Draw a border around the playing field
- Different types of enemies with different algorithms to find you.
- Different color for enemies and player