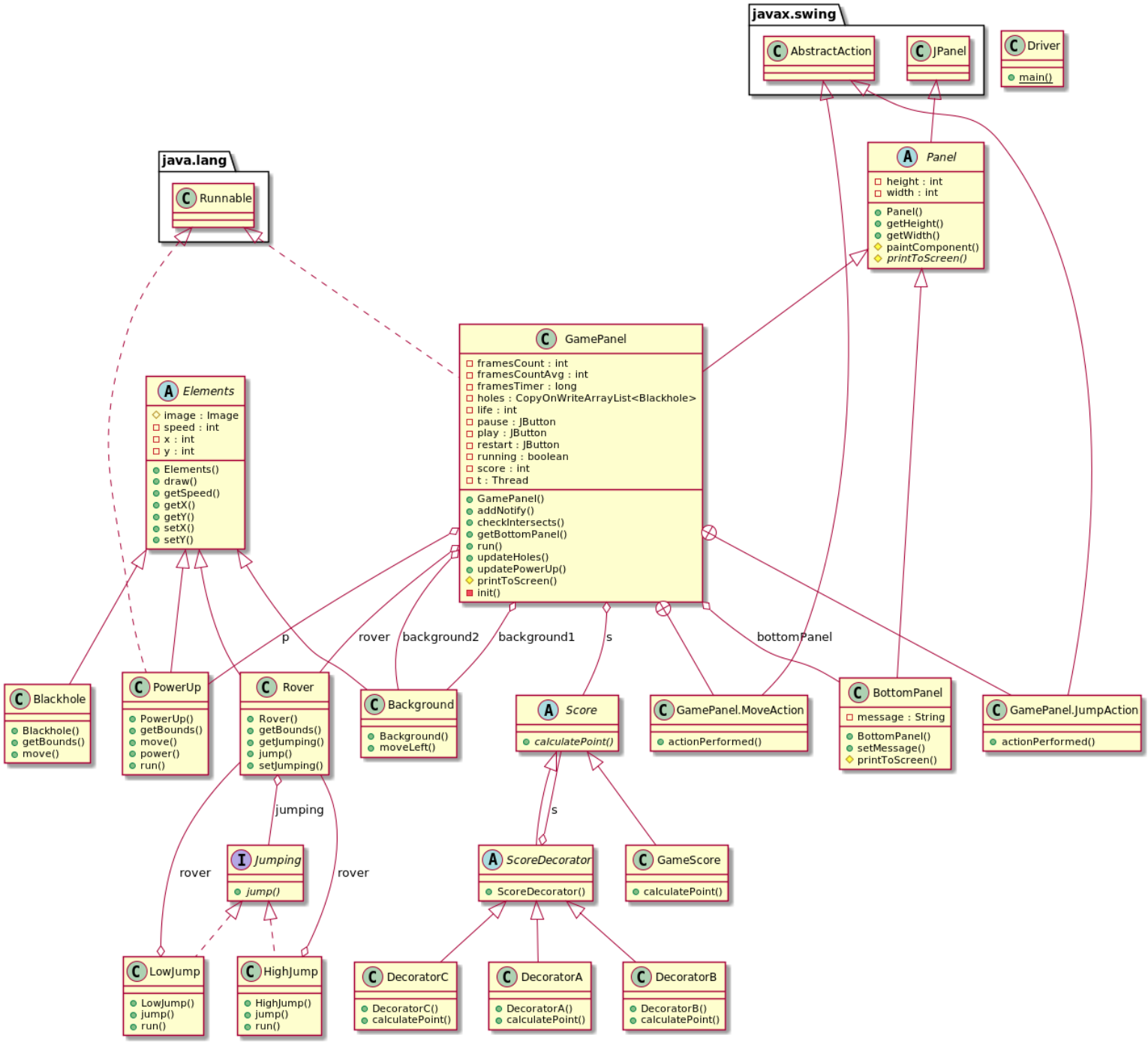
SRC's Class Diagram



Strategy Pattern : To be able to change the character's jumping behavior dynamically and be able to add new jumping styles, a Jumping interface with the jump method had been created. Rover class (Character) uses its jumping object's jump method to be able to jump over the obstacles in the game. Initially jumping object is an instance of LowJump class which implements the Jumping interface and overrides the jump method to execute low jump behavior. If the rover gets power-up, the jumping object is assigned to HighJump instance which also implements the Jumping interface and overrides the jump method to execute high jump behavior in run time.

Decorator Pattern : To implement decorator pattern, abstract Score class which has abstract calculatePoint method is used as a component and GameScore class is used as a concrete component. Abstract Decorator class, ScoreDecorator, has a Score object to cover the current Score instance. It is also used to be able to add additional power-ups. Decorator A, B, C classes which are representative of the power-ups extend the ScoreDecorator. By default, Score is a GameScore instance that calculates the default points that are earned when the rover jumps over the obstacles.

When the rover acquires a power-up, a decorator class covers the score object to attach additional responsibilities to the object. For example, if Decorator A covers the Game Score, calculatePoint method returns default points * 2.

Except for design patterns, 2 abstract classes were created, first one is the Elements class which is the supertype for elements (Rover , Background, Blackhole, PowerUp) that will be appeared in the game panel. Most of the elements also implement the Runnable interface to use threads. The second one is the Panel class which implements Jpanel to be able to add panels to Jframe. This class is the supertype for GamePanel and BottomPanel. The game panel also implements the Runnable interface to use threads for the gaming loop.