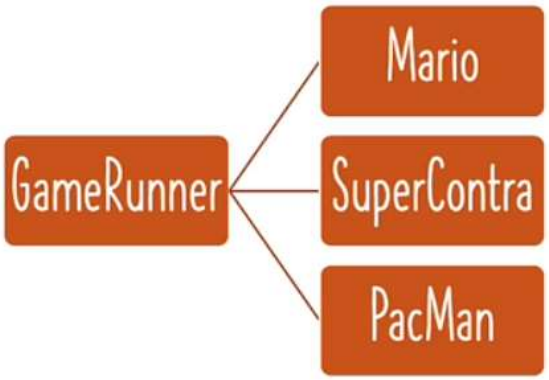
INTRODUCING JAVA INTERFACE TO MAKE APPLICATION LOOSELY COUPLED

Let's get start with:

2. **Iteration 2**: **Loose Coupling - Interfaces**
    - GameRunner class
    - GamingConsole interface
        - Game classes: Mario, SuperContra, Pacman etc

Till now we have, GameRunner which runs a specific game.



The **GameRunner** should **interact** with **GamingConsole** interface. With this, we can **decouple** the **GameRunner** with a **specific game**.

We know the MarioGame, SuperContraGame, PacManGame have the same or common actions or method like up(), down(), left(), right().

INTERFACES: An interface is something which represents the common actions that can be performed on a specific set of classes.

Up(), down(), left() and right() these are the four actions that we would want to be able to perform using **GamingConsole** interface. These four actions already implemented in MarioGame and SuperContraGame.

GamingConsole.java (Interface)

```
package com.naveen.learnspringframework.game;

public interface GamingConsole {
    void up();
    void down();
    void left();
    void right();
}
```

Now all the games should implement the GamingConsole interface.

SuperContraGame.java

```
package com.naveen.learnspringframework.game;

public class SuperContraGame implements GamingConsole {
    public void up() {
        System.out.println("up");
    }

    public void down() {
        System.out.println("Sit down");
    }

    public void left() {
        System.out.println("Go back");
    }

    public void right() {
        System.out.println("Shoot a bullet");
    }
}
```

We wouldn't get any error, because we **already implemented** all the actions which is present in the **GamingConsole** interface.

MarioGame.java

```
package com.naveen.learnspringframework.game;

public class MarioGame implements GamingConsole {

    public void up() {
        System.out.println("Jump");
    }

    public void down() {
        System.out.println("Go into a hole");
    }

    public void left() {
        System.out.println("Go back");
    }

    public void right() {
        System.out.println("Accelerate");
    }
}
```

PacmanGame.java

```
package com.naveen.learnspringframework.game;

public class PacmanGame implements GamingConsole {
    public void up() {
        System.out.println("up");
    }

    public void down() {
        System.out.println("down");
    }

    public void left() {
        System.out.println("left");
    }

    public void right() {
        System.out.println("right");
    }
}
```

But our **GameRunner class is still tightly coupled** to a specific game. So, instead of make using a specific game, will make use of the **GamingConsole** interface.

GameRunner.java

```
package com.naveen.learnspringframework.game;

public class GameRunner {
    private GamingConsole game;

    public GameRunner(GamingConsole game) {
        this.game = game;
    }

    public void run() {
        System.out.println("Running game: " + game);
        game.up();
        game.down();
        game.left();
        game.right();
    }
}
```

All the method calls would still work because these methods are part of the **GamingConsole** interface definition.

AppGamingBasicJava.java

```
package com.naveen.learnspringframework;

import com.naveen.learnspringframework.game.GameRunner;
import com.naveen.learnspringframework.game.MarioGame;
import com.naveen.learnspringframework.game.SuperContraGame;

public class AppGamingBasicJava {

    public static void main(String[] args) {

        //var game = new MarioGame();
        var game = new SuperContraGame();
        var gameRunner = new GameRunner(game);
        gameRunner.run();
    }
}
```

OUTPUT:

```
Running game: com.naveen.learnspringframework.game.SuperContraGame@1c4af82c
up
Sit down
Go back
Shoot a bullet
```

AppGamingBasicJava.java

```java
package com.naveen.learnspringframework;

import com.naveen.learnspringframework.game.GameRunner;
import com.naveen.learnspringframework.game.MarioGame;
import com.naveen.learnspringframework.game.PacmanGame;
import com.naveen.learnspringframework.game.SuperContraGame;

public class AppGamingBasicJava {

    public static void main(String[] args) {

        //var game = new PacmanGame();
        var game = new MarioGame();
        //var game = new SuperContraGame();
        var gameRunner = new GameRunner(game);
        gameRunner.run();
    }
}
```

OUTPUT:

```
Running game: com.naveen.learnspringframework.game.MarioGame@1c4af82c
Jump
Go into a hole
Go back
Accelerate
```

We **don't** need to make any code change in **GameRunner** class when we want to switch from one game to another.

Once we create an interface and make our game classes implement the interface, we can make use of the interface from the **GameRunner** class. So, we have **decoupled or loosely coupled** our **GameRunner** class with a specific game using **GamingConsole** interface. It doesn't matter which game we are running. We **don't** need to make any code change in **GameRunner.**