

# Inversión de control y Hotspots

templates y hooks  
herencia y composición

# Antes de seguir...

- Template method
- Frameworks (reuso)
- Refactoring to Patterns (from Template)

# Material adicional



## Plantillas y ganchos

### TOC

- Plantillas y ganchos
  - Preliminares
    - Conocimientos previos
    - Lectura previa recomendada
  - Ejemplo conductor
  - Diseño general
  - Plantillas y herencia
    - Ejercicio - ejemplo
  - Plantillas y composición
    - Ejercicio - ejemplo
  - Recapitulando
  - Auto-evaluación
  - Lectura recomendada

### Plantillas y ganchos

#### Objetivos

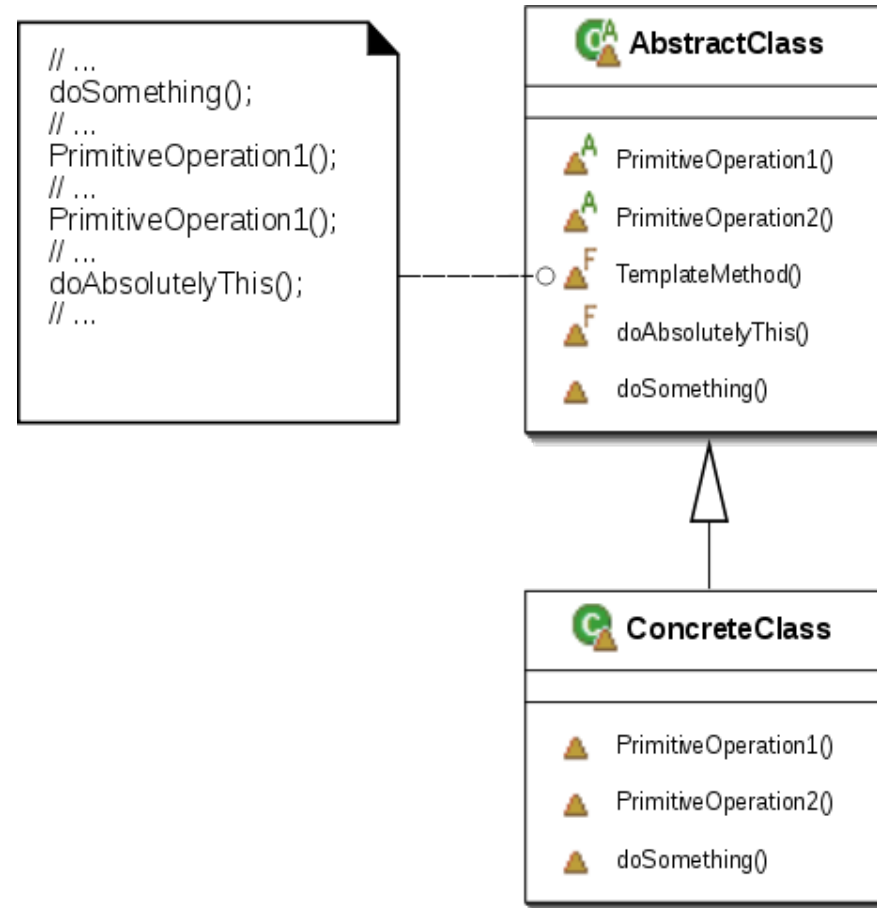
Las plantillas y los ganchos son una estrategia de programación comúnmente utilizada para introducir puntos de variabilidad (hotspots) en los frameworks orientados a objetos. Pueden implementarse utilizando herencia o composición como mecanismos para separar los aspectos comunes de los aspectos variables. El objetivo de este material es que usted pueda reconocer ambas estrategias de implementación del concepto de plantilla y ganchos.

```
classDiagram
    class CamelBoard {
        run()
        addRobot(Robot : r)
    }
    class Robot {
        <<abstract>>
        <<template method>> step()
        <<abstractOp>> move()
        <<abstractOp>> consumeBattery()
        <<abstractOp>> fireArms()
        <<hookOp>> collectArtifacts()
    }
    class SolarRobot {
        <<abstract>>
    }
    class NuclearRobot {
        <<abstract>>
    }
    CamelBoard "1" -- "1..*" Robot : robots
    Robot <|-- SolarRobot
    Robot <|-- NuclearRobot
```

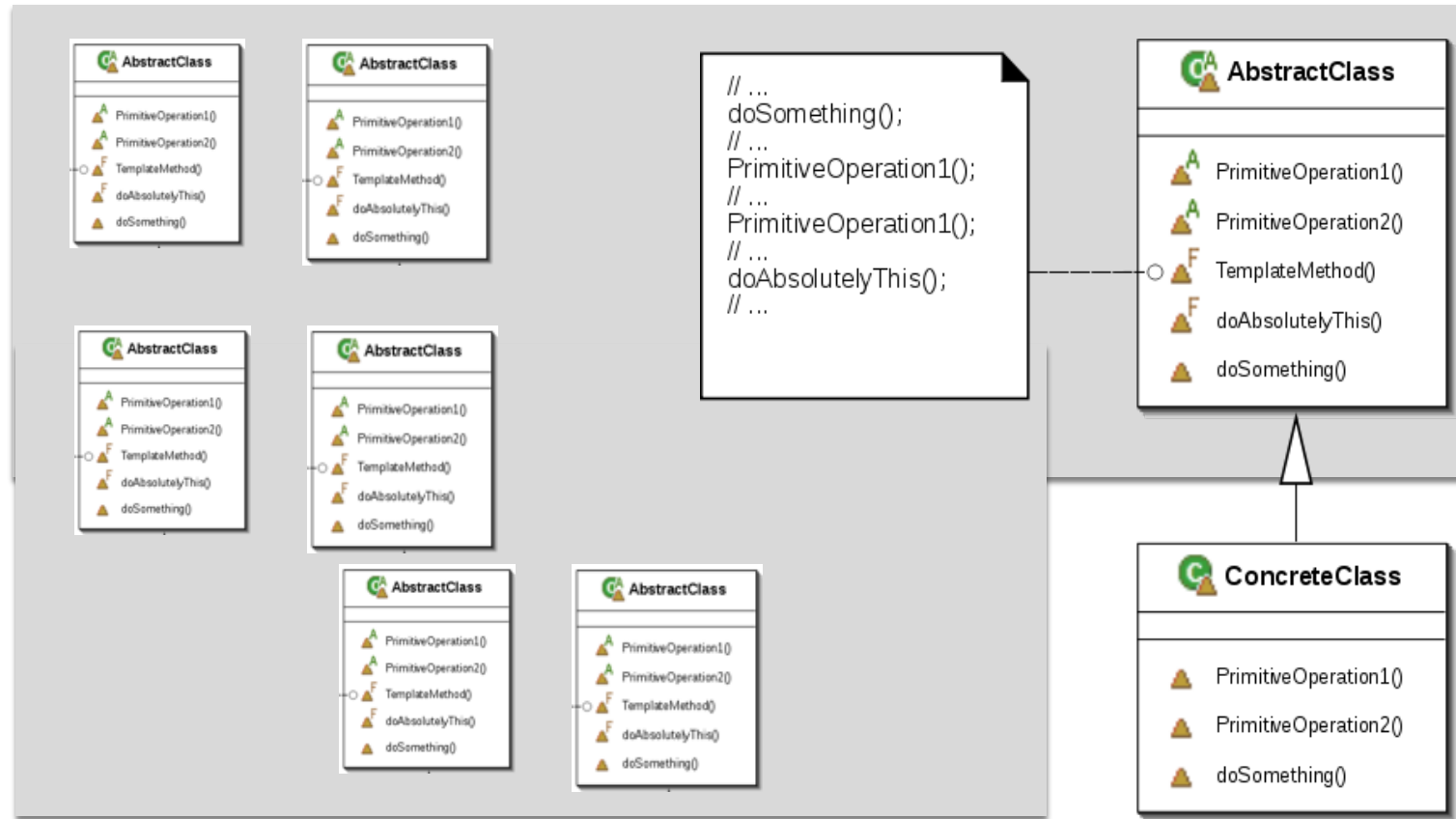
# Inversión de control

- La inversión de control es la característica principal de la arquitectura run-time de un framework
- Esta característica permite que los pasos canónicos de procesamiento de la aplicación (comunes a todas las instancias) sean especializados por **objetos tipo manejadores de eventos** a los que invoca el framework como parte de su mecanismo reactivo de despacho

# Template method



# Template method en los puntos de extensión



# Robot wars - Framework



# Robot wars - Framework

## Frozen spot

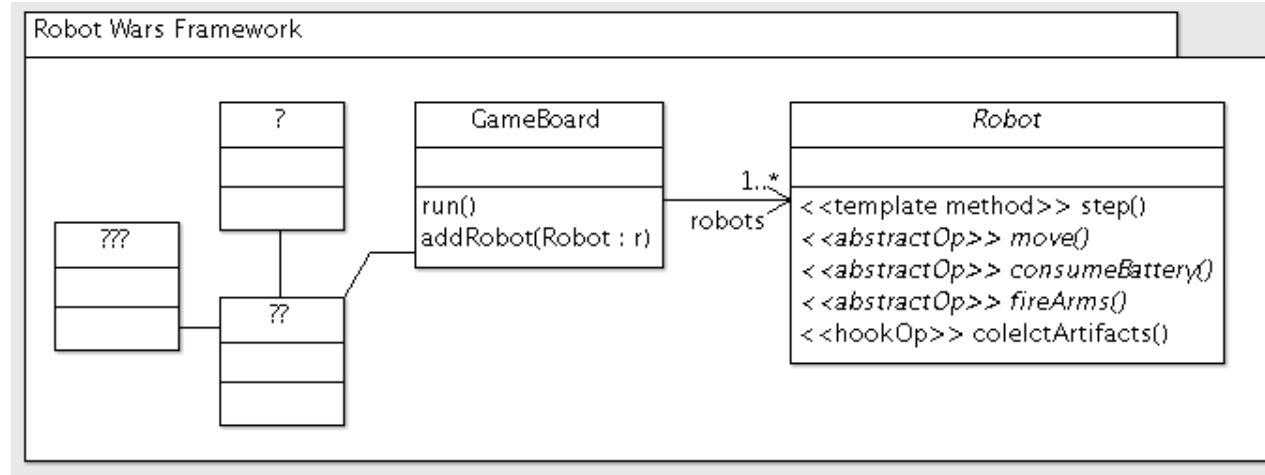
- En cada ronda, se activan los robots uno a uno - reciben el mensaje **step()**
- Ejecutan las siguientes operaciones en orden:
  - Avanzar
  - Consumir batería
  - Disparar
  - Recolectar artefactos

## Hot spots

- Varias formas de avanzar:
  - Orugas, Ruedas, Overcraft
- Varios tipos de baterías:
  - Cadmio, Nuclear, Solar
- Varios equipos de armamento:
  - Laser, Bombas, ...

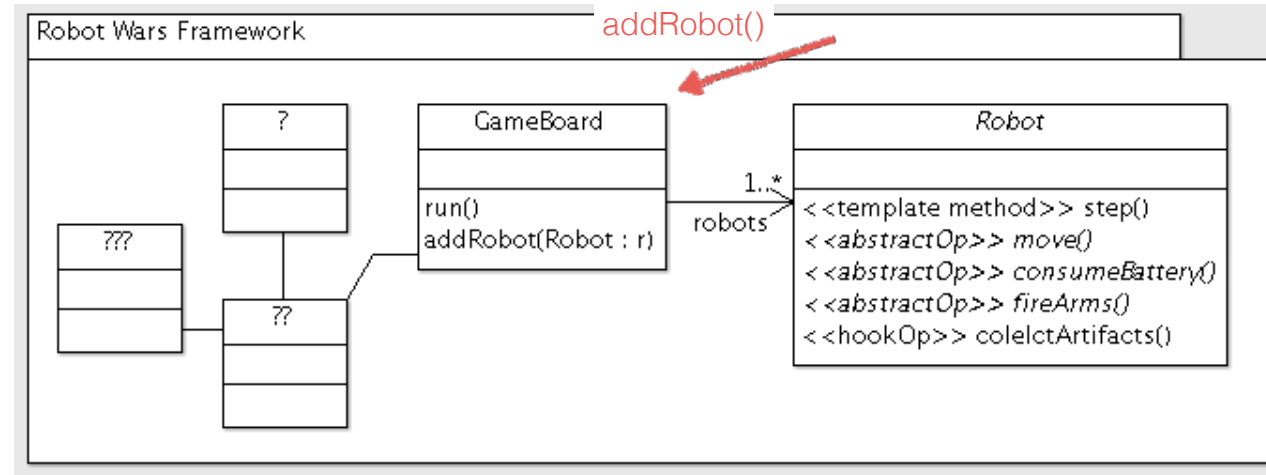


# Frozen/Hot spots



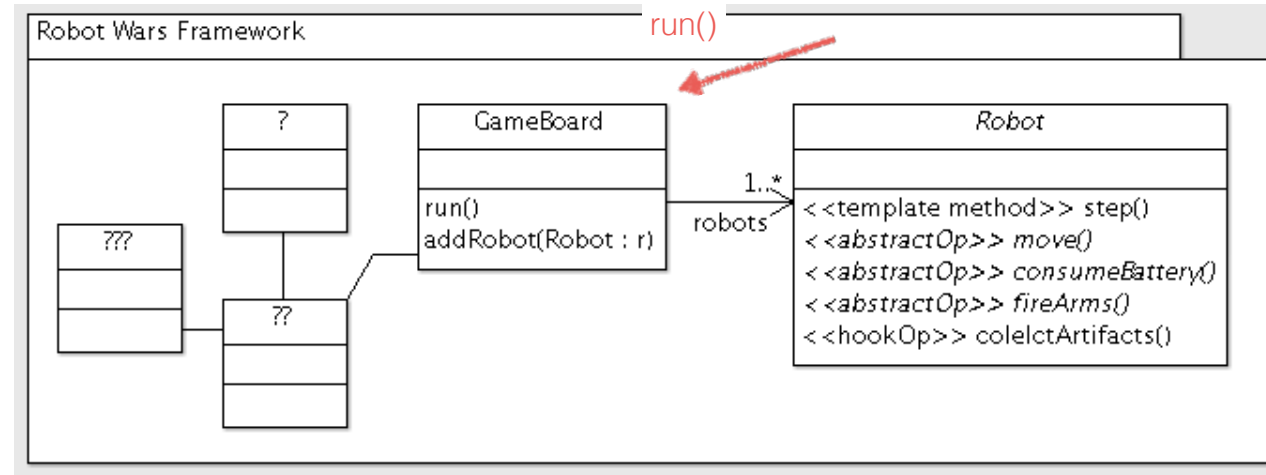
- Agregamos robots uno a uno con `addRobot()`
- Iniciamos el juego con `run()` o `runForCycles(n)`
- El frozenspot oculta el loop de control
- Periodicamente envía `step()` a todos los robots
- El template garantiza que se cumple el orden en los pasos

# Frozen/Hot spots



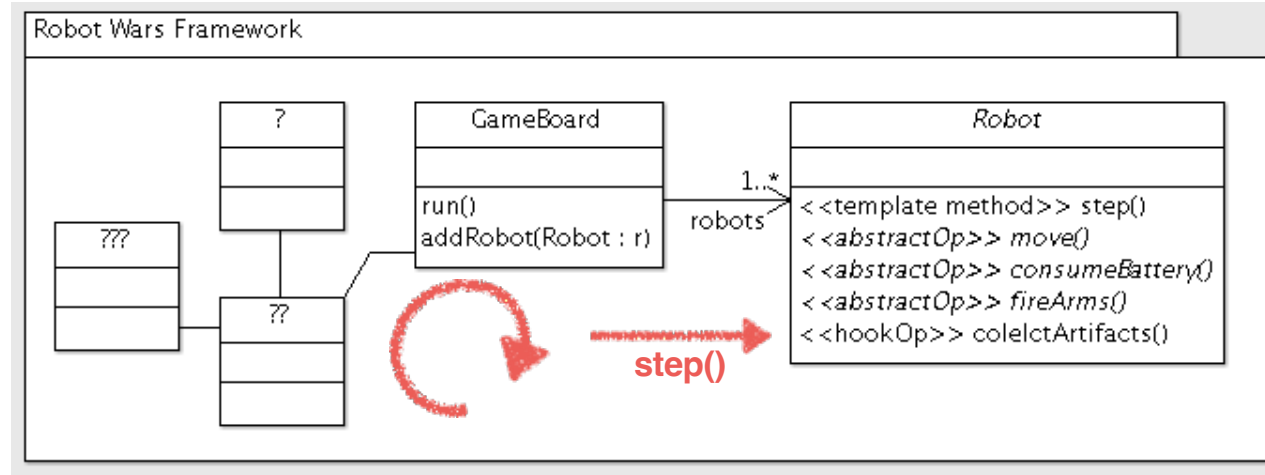
- Agregamos robots uno a uno con `addRobot()`
- Iniciamos el juego con `run()` o `runForCycles(n)`
- El frozenspot oculta el loop de control
- Periodicamente envía `step()` a todos los robots
- El template garantiza que se cumple el orden en los pasos

# Frozen/Hot spots

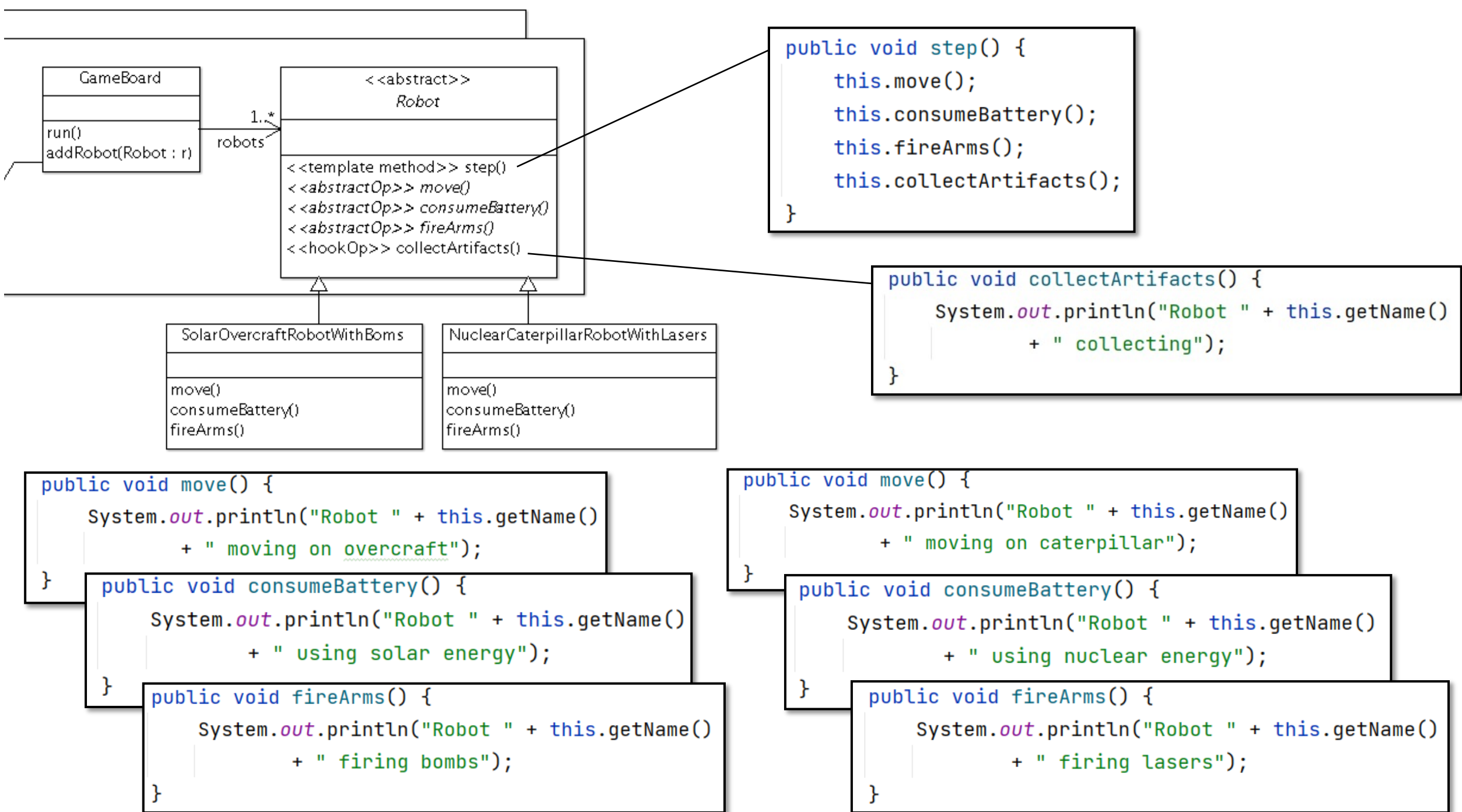


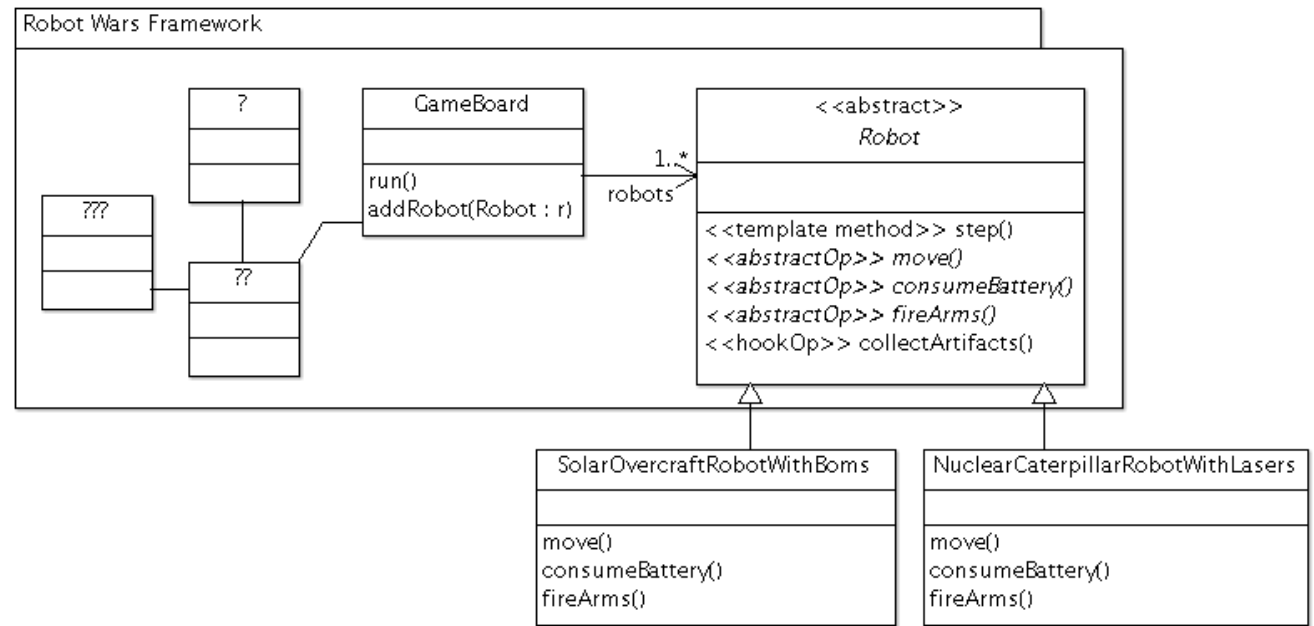
- Agregamos robots uno a uno con `addRobot()`
- Iniciamos el juego con `run()` o `runForCycles(n)`
- El frozenspot oculta el loop de control
- Periodicamente envía `step()` a todos los robots
- El template garantiza que se cumple el orden en los pasos

# Frozen/Hot spots



- Agregamos robots uno a uno con `addRobot()`
- Iniciamos el juego con `run()` o `runForCycles(n)`
- El frozenspot oculta el loop de control
- Periodicamente envía `step()` a todos los robots
- El template garantiza que se cumple el orden en los pasos





```

public class SimpleGame {
    public static void main(String[] args) {
        GameBoard board = new GameBoard();
        board.add(new NuclearCaterpillarRobotWithLasers("Twonky"));
        board.add(new SolarOvercraftRobotWithBoms("Hammer Bot"));
        board.runForCicles(5);
    }
}
  
```

# ¿Como serán?

SolarOvercraftRobotWithLasers

SolarCaterpillarRobotWithLasers

....

¿cómo lo resolvemos?

- Varias formas de avanzar: Orugas, Ruedas, Overcraft
- Varios tipos de baterías: Cadmio, Nuclear, Solar
- Varios equipos de armamento: Laser, Bombas, ...

SolarOvercraftRobotWithBoms
move() consumeBattery() fireArms()

NuclearCaterpillarRobotWithLasers
move() consumeBattery() fireArms()

```
public void move() {  
    System.out.println("Robot " + this.getName()  
        + " moving on overcraft");  
}
```

```
public void consumeBattery() {  
    System.out.println("Robot " + this.getName()  
        + " using solar energy");  
}
```

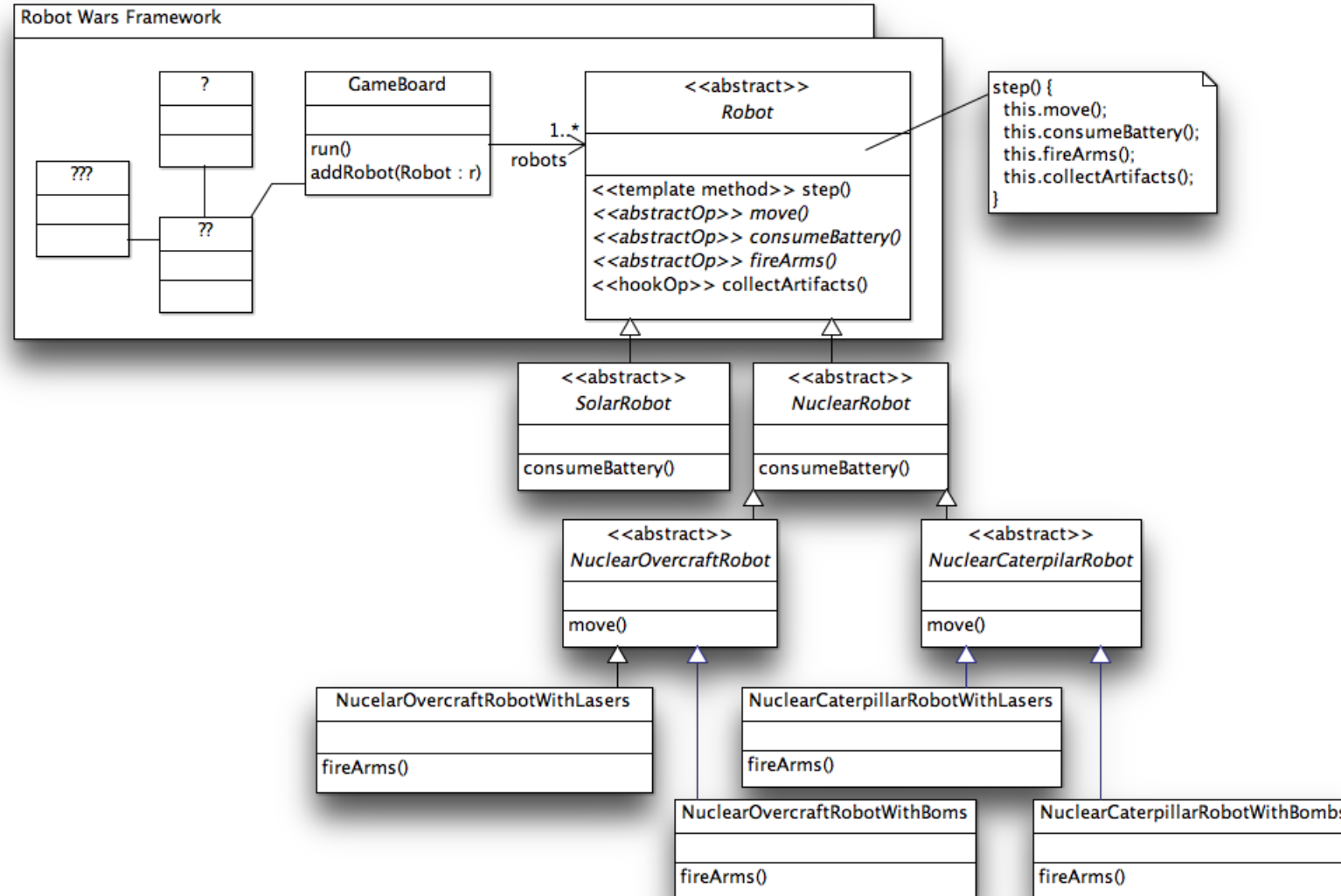
```
public void fireArms() {  
    System.out.println("Robot " + this.getName()  
        + " firing bombs");  
}
```

```
public void move() {  
    System.out.println("Robot " + this.getName()  
        + " moving on caterpillar");  
}
```

```
public void consumeBattery() {  
    System.out.println("Robot " + this.getName()  
        + " using nuclear energy");  
}
```

```
public void fireArms() {  
    System.out.println("Robot " + this.getName()  
        + " firing lasers");  
}
```

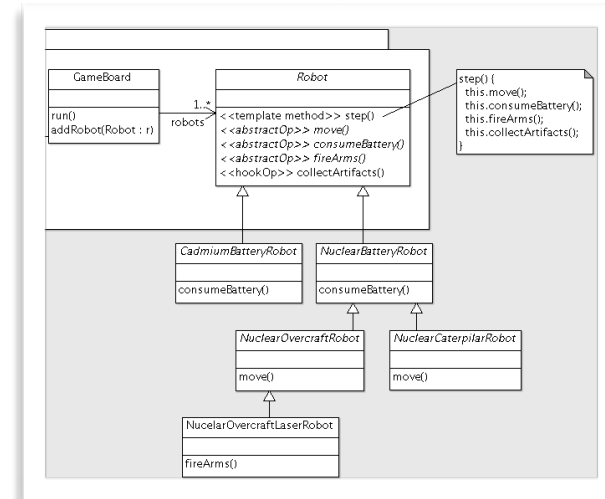
# Hot spots con herencia





# Hot spots con herencia

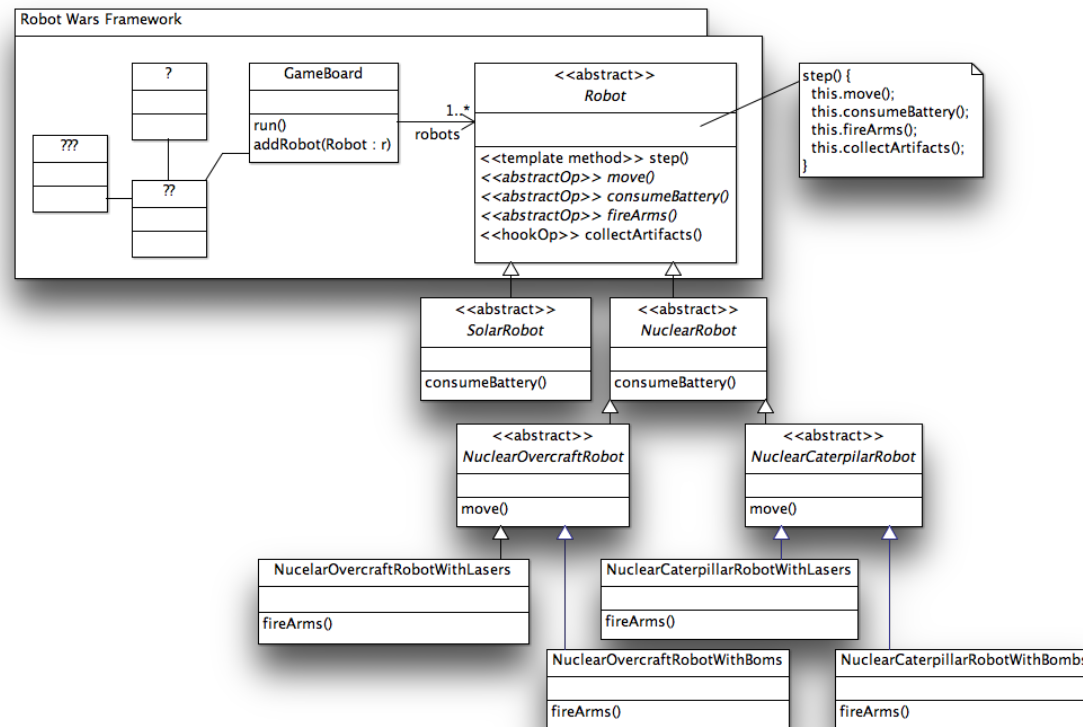
- El template method asegura que se cumplan los pasos
- Programamos “por diferencia” implementando los hooks, utilizando herencia
- Tenemos acceso a las V.I. y métodos que heredamos (muy flexible)
- PERO, nuestra jerarquía explota si hay muchas combinaciones



- No podemos cambiar el comportamiento del Robot en run-time

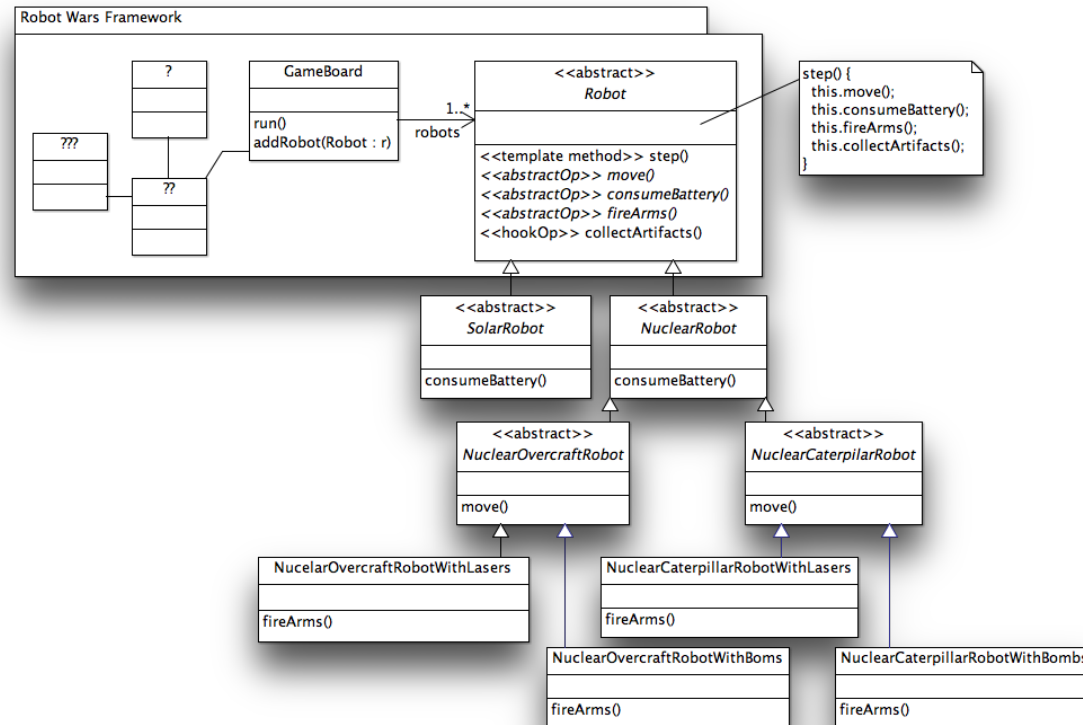
# Hot spots con herencia

- ¿Que tenemos que hacer si aparece una nueva alternativa para algún componente del Robot?



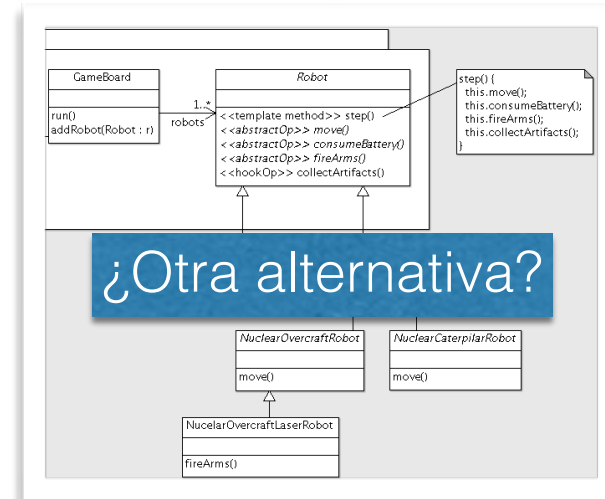
# Hot spots con herencia

- ¿Una vez instanciado el Robot, puedo cambiarle un componente?



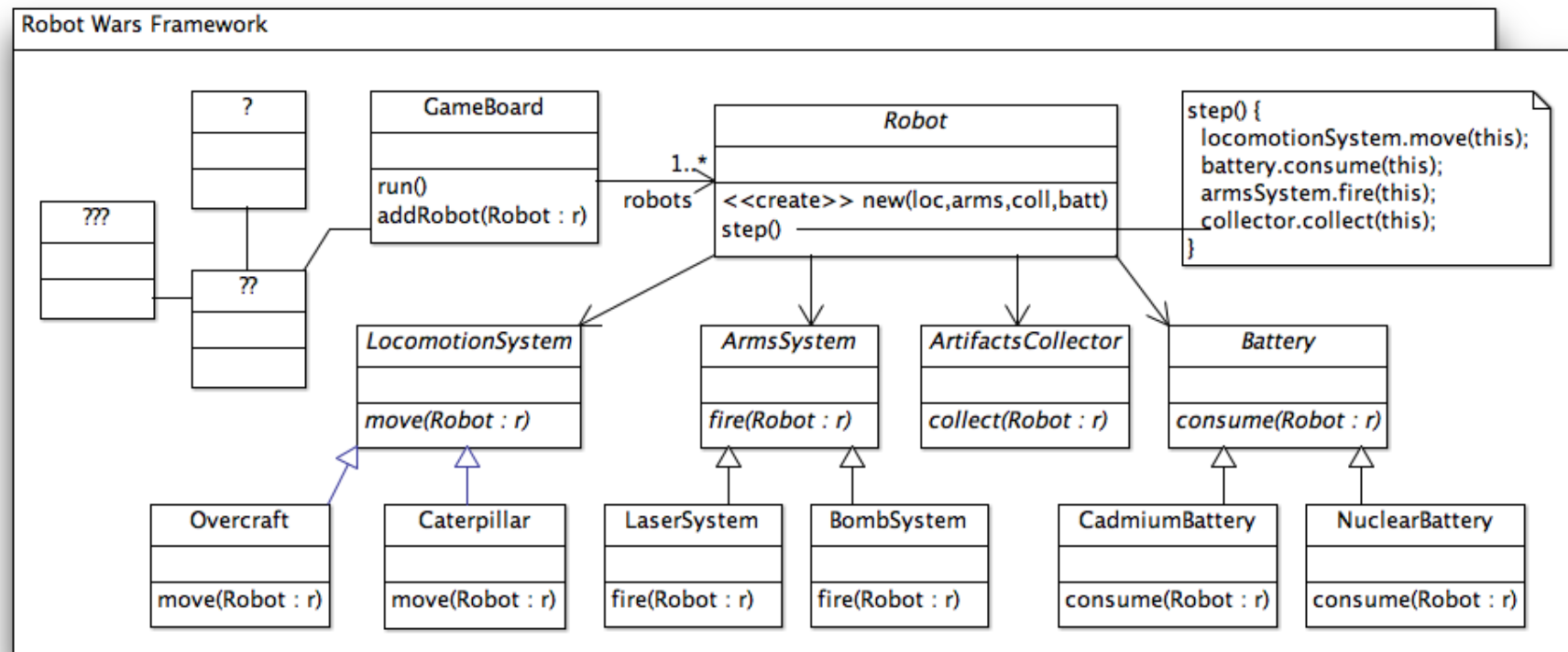
# Hot spots con herencia

- El template method asegura que se cumplan los pasos
- Programamos “por diferencia” implementando los hooks, utilizando herencia
- Tenemos acceso a las V.I. y métodos que heredamos (muy flexible)
- PERO, nuestra jerarquía explota si hay muchas combinaciones



- No podemos cambiar el comportamiento del Robot en run-time

# Hotspots con composición

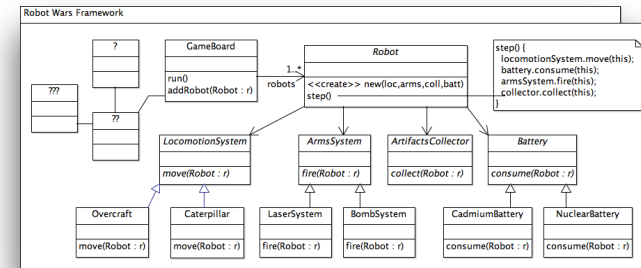


# Hotspots con composición

```
public class SimpleGame {  
    public static void main(String[] args) throws IOException {  
        GameBoard board = new GameBoard();  
        board.add(new Robot("Twonky", new Caterpillar(),  
                             new NuclearPlant(), new BombsSystem()));  
        board.add(new Robot("Hammer Bot", new Overcraft(),  
                             new NuclearPlant(), new LasersSystem()));  
        board.runForCicles(5);  
    }  
}
```

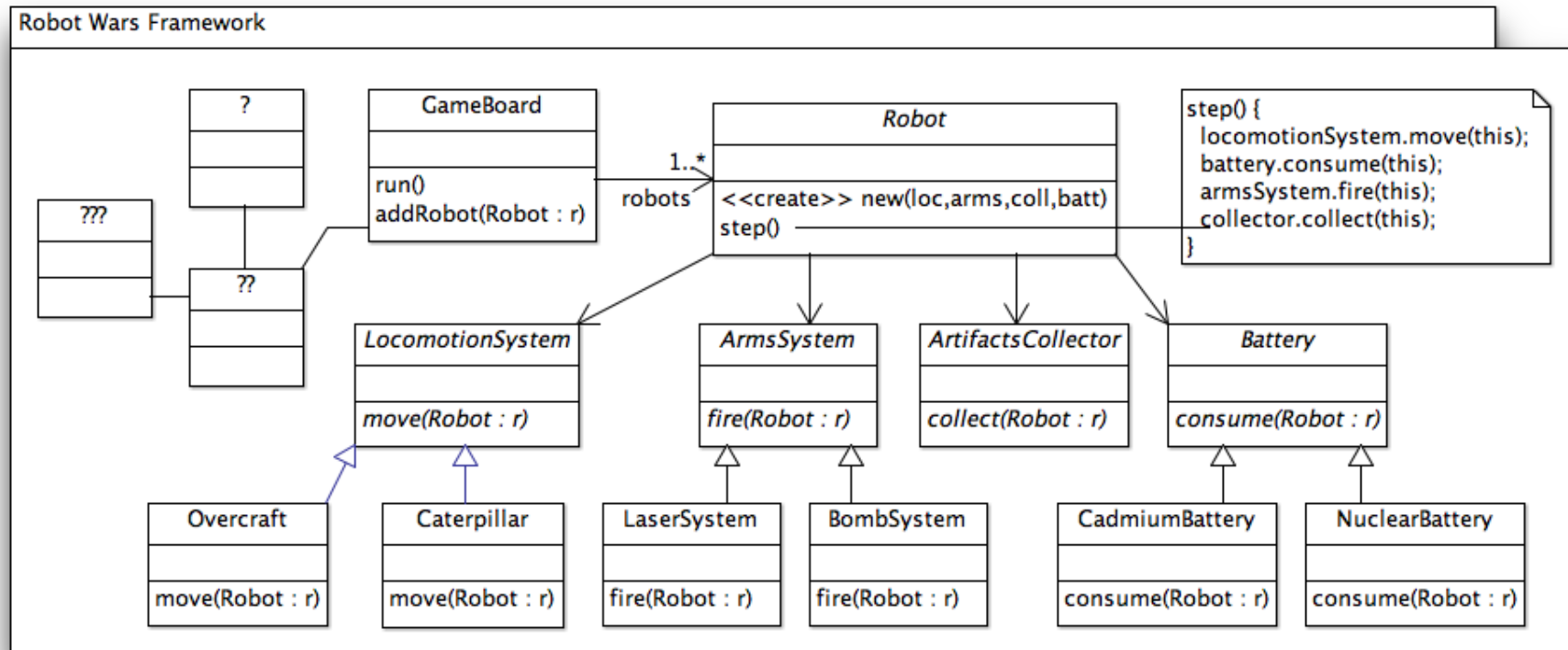
# Hot spots con composición

- El template - step() - asegura que se cumplan los pasos
- Implementamos los hooks, utilizando composición
- NO tenemos acceso a las V.I. ni métodos privados de Robot ni de los otros componentes.
- Dependemos de las opciones provistas
- “Podríamos” sub-clasificar las componentes



# Hotspots con composición

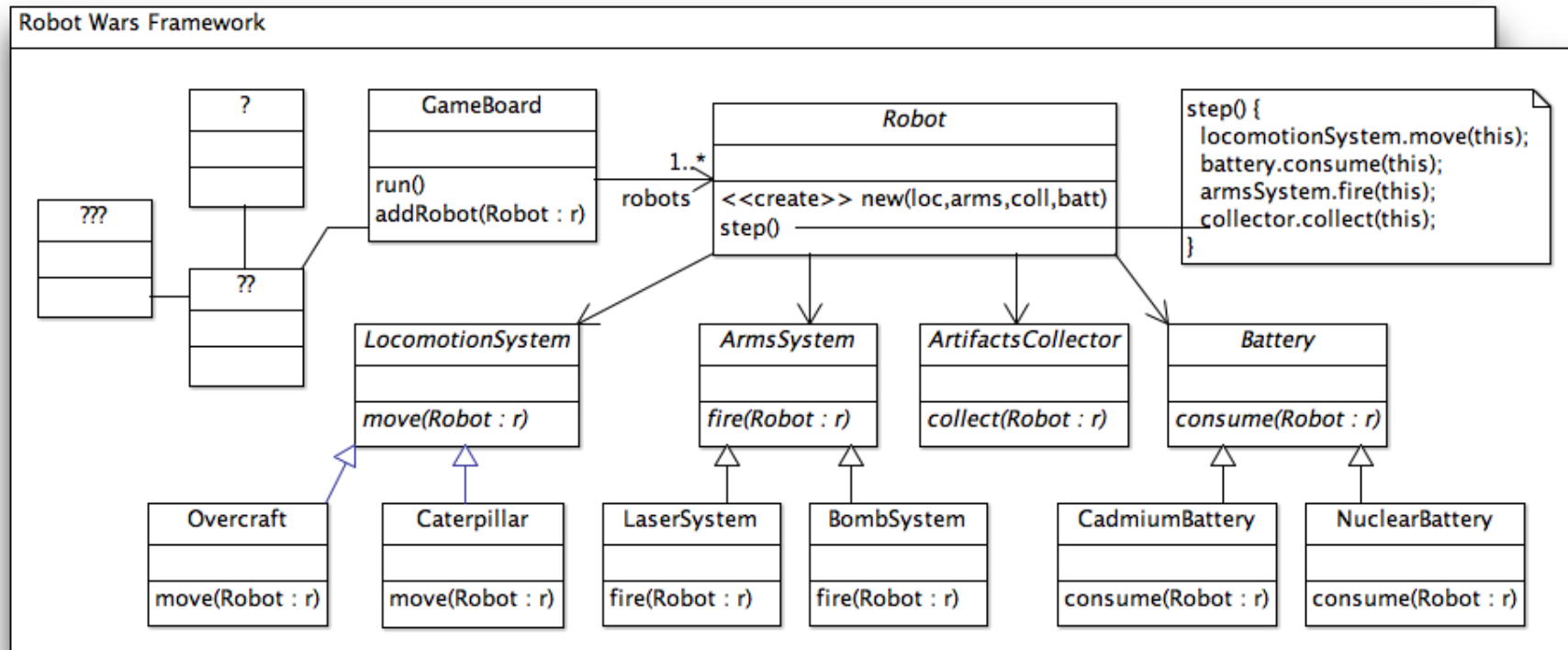
- ¿Que tenemos que hacer si aparece una nueva alternativa para algún componente del Robot?





# Hotspots con composición

- ¿Una vez instanciado el Robot, puedo cambiarle un componente?



# Material adicional



## Plantillas y ganchos

### TOC

- Plantillas y ganchos
  - Preliminares
    - Conocimientos previos
    - Lectura previa recomendada
  - Ejemplo conductor
  - Diseño general
  - Plantillas y herencia
    - Ejercicio - ejemplo
  - Plantillas y composición
    - Ejercicio - ejemplo
  - Recapitulando
  - Auto-evaluación
  - Lectura recomendada

### Plantillas y ganchos

#### Objetivos

Las plantillas y los ganchos son una estrategia de programación comúnmente utilizada para introducir puntos de variabilidad (hotspots) en los frameworks orientados a objetos. Pueden implementarse utilizando herencia o composición como mecanismos para separar los aspectos comunes de los aspectos variables. El objetivo de este material es que usted pueda reconocer ambas estrategias de implementación del concepto de plantilla y ganchos.