

# Inversión de control y Hotspots

templates y hooks  
herencia y composición

# Antes de seguir...

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

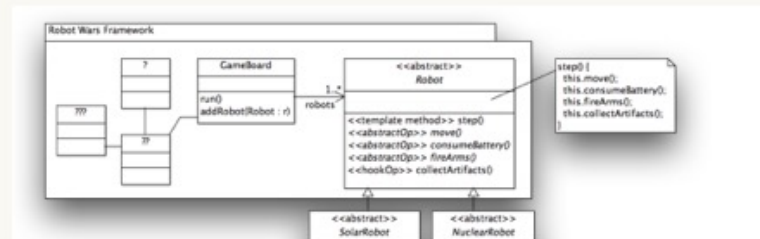
<< < ^ > >>

## Plantillas y ganchos



### Objetivos

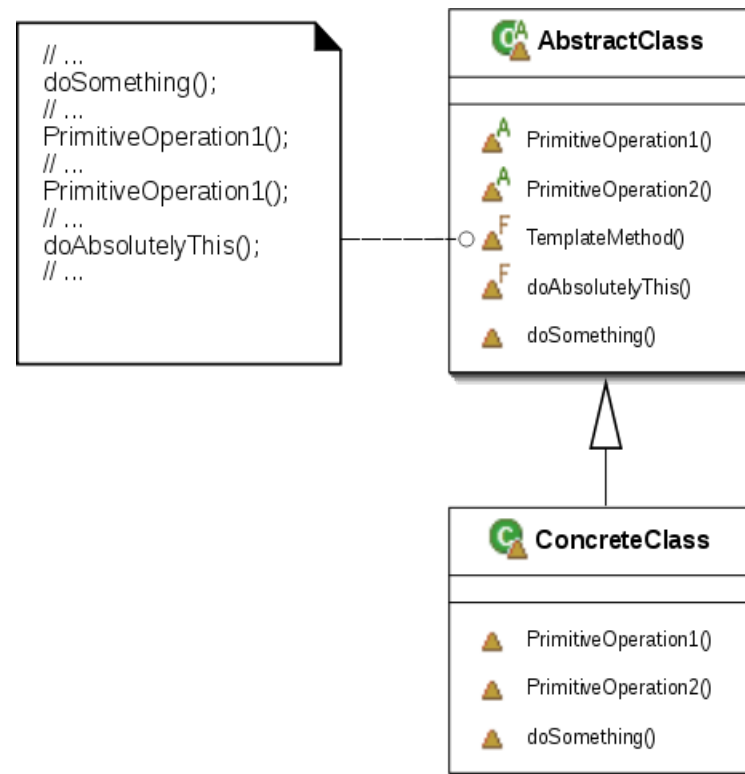
Las plantillas y los ganchos son una estrategia de programación comúnmente utilizada para introducir puntos de variabilidad (hotspots) en los frameworks orientados a objetos. Pueden implementarse utilizando herencia o composición como mecanismos para separar los aspectos comunes de los aspectos variables.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



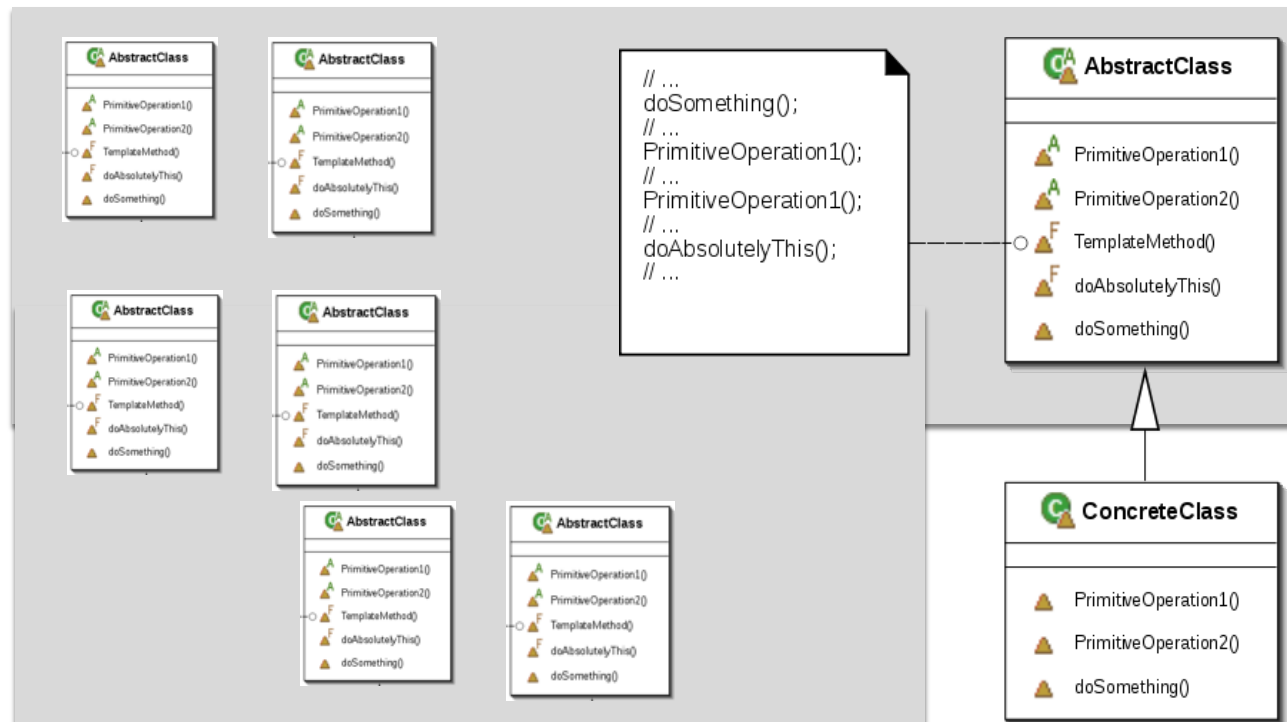
# Inversión de control

- 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

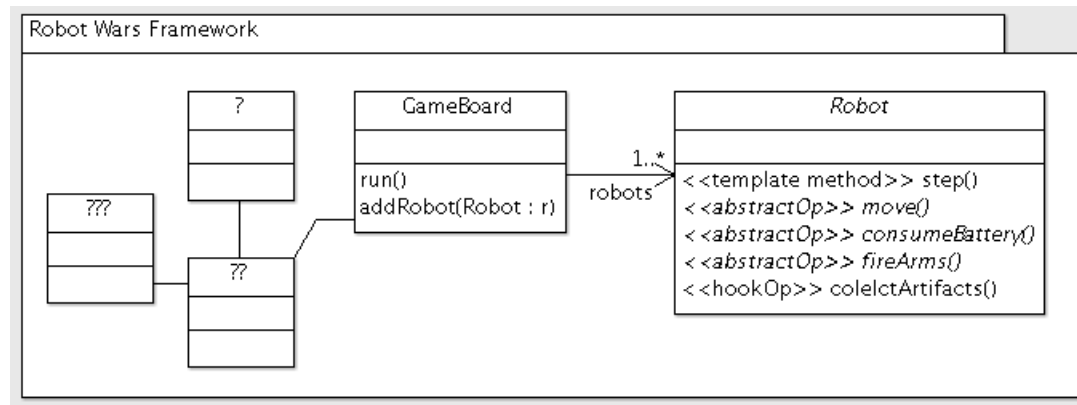
- En cada ronda, se activan los robots uno a uno - reciben el mensaje **step()**
- Ejecutan las siguientes operaciones en orden:
  - Avanzar
  - Consumir batería
  - Disparar
  - Recolectar artefactos

## Hot spots

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

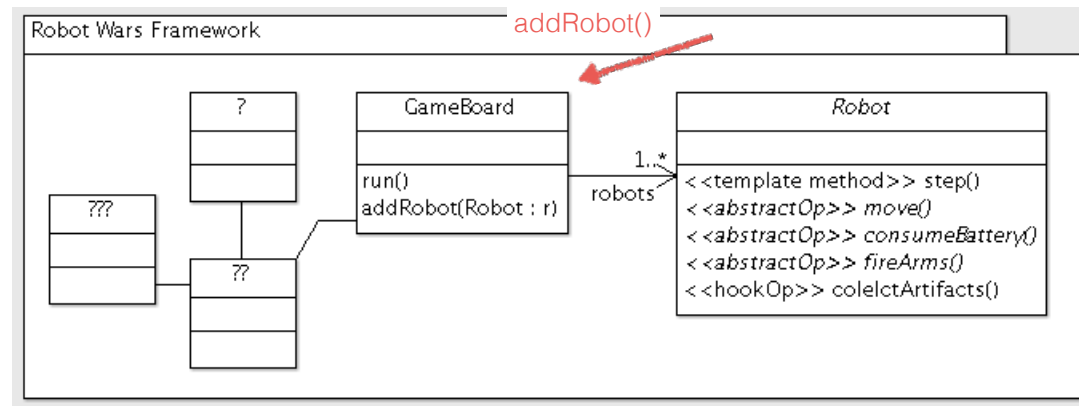


# Frozen/Hot spots



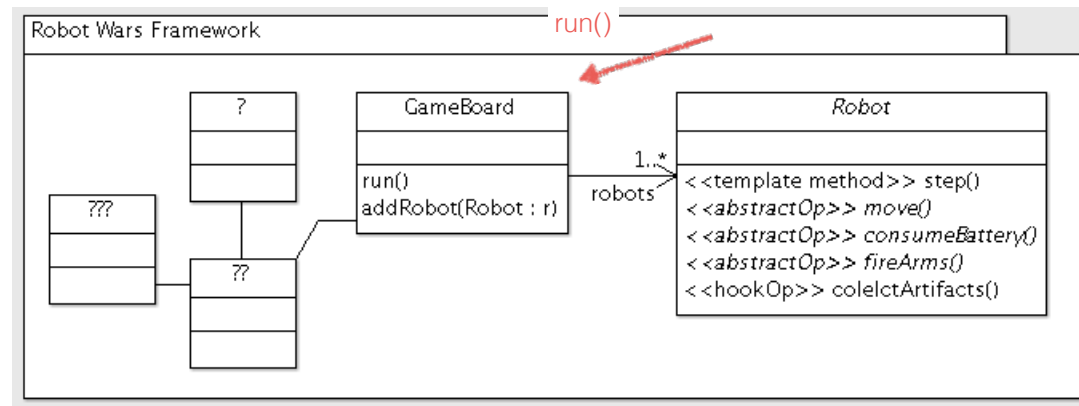
- Agregamos robots uno a uno con `addRobot()`
- Iniciamos el juego con `run()` o `runForCycles(n)`
- El frozenspot oculta el loop de control
- Periodicamente envía `step()` a todos los robots
- El template garantiza que se cumple el orden en los pasos

# Frozen/Hot spots



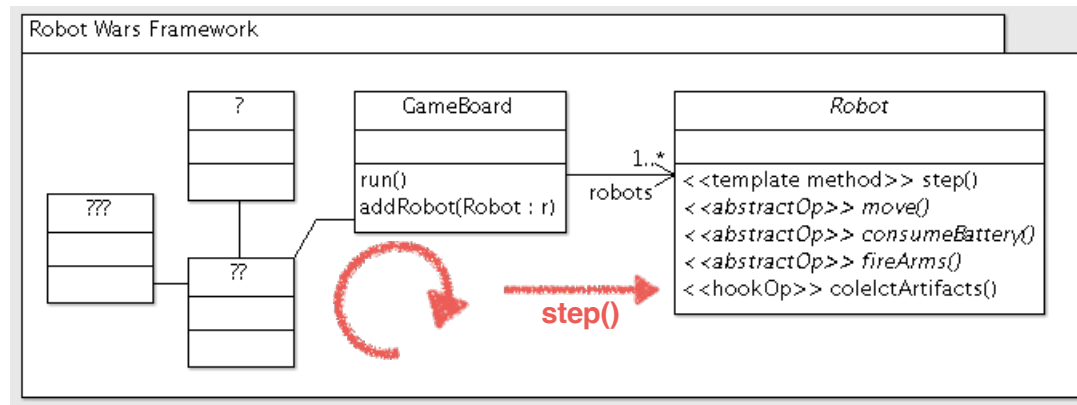
- Agregamos robots uno a uno con `addRobot()`
- Iniciamos el juego con `run()` o `runForCycles(n)`
- El frozenspot oculta el loop de control
- Periodicamente envía `step()` a todos los robots
- El template garantiza que se cumple el orden en los pasos

# Frozen/Hot spots

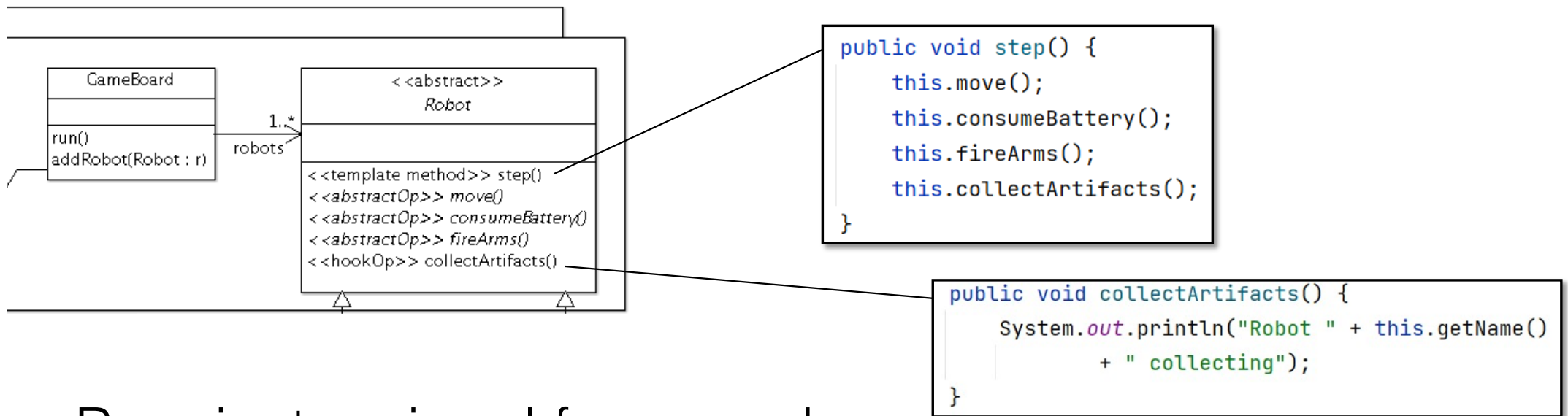


- Agregamos robots uno a uno con `addRobot()`
- Iniciamos el juego con `run()` o `runForCycles(n)`
- El frozenspot oculta el loop de control
- Periodicamente envía `step()` a todos los robots
- El template garantiza que se cumple el orden en los pasos

# Frozen/Hot spots



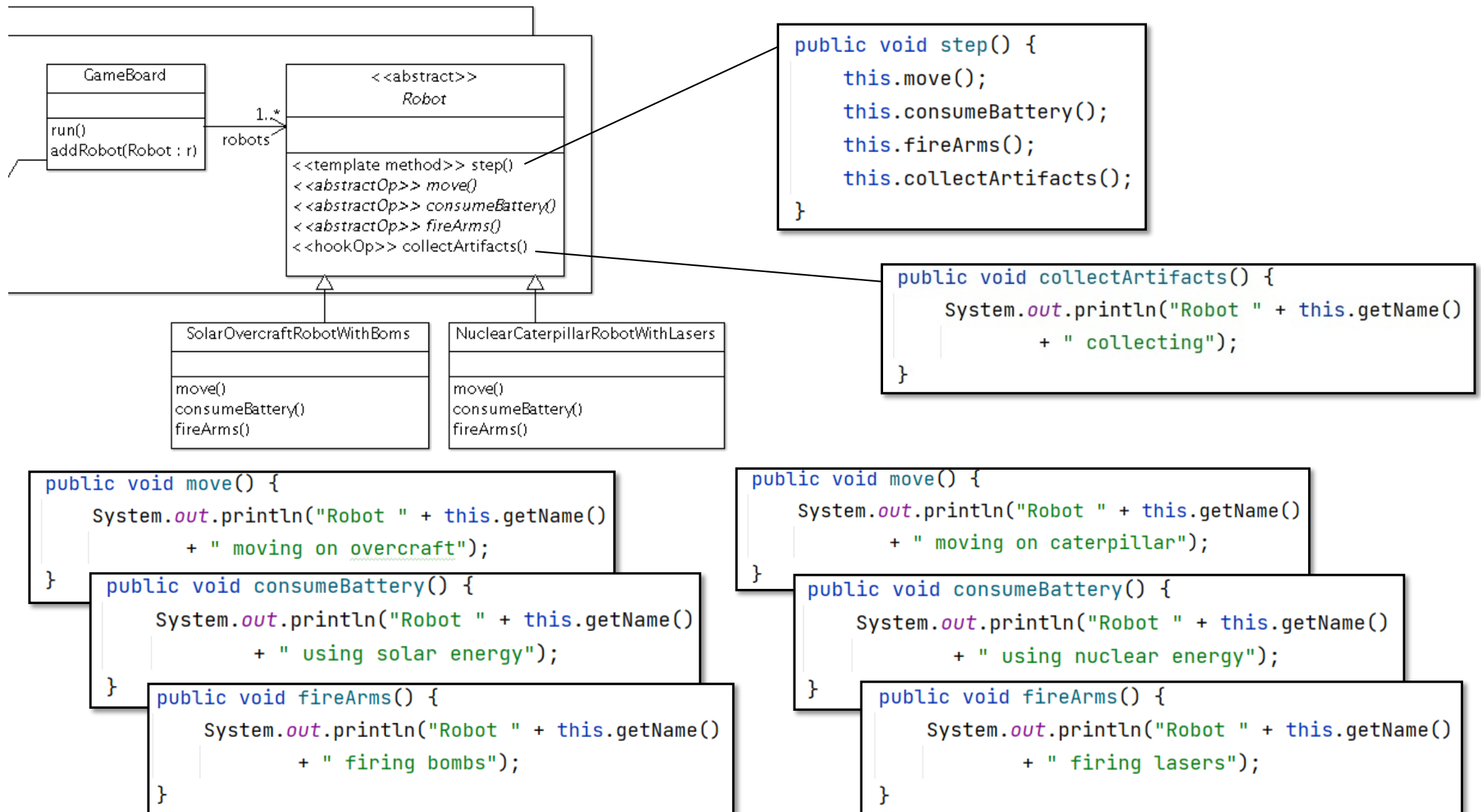
- Agregamos robots uno a uno con `addRobot()`
- Iniciamos el juego con `run()` o `runForCycles(n)`
- El frozenspot oculta el loop de control
- Periodicamente envía `step()` a todos los robots
- El template garantiza que se cumple el orden en los pasos

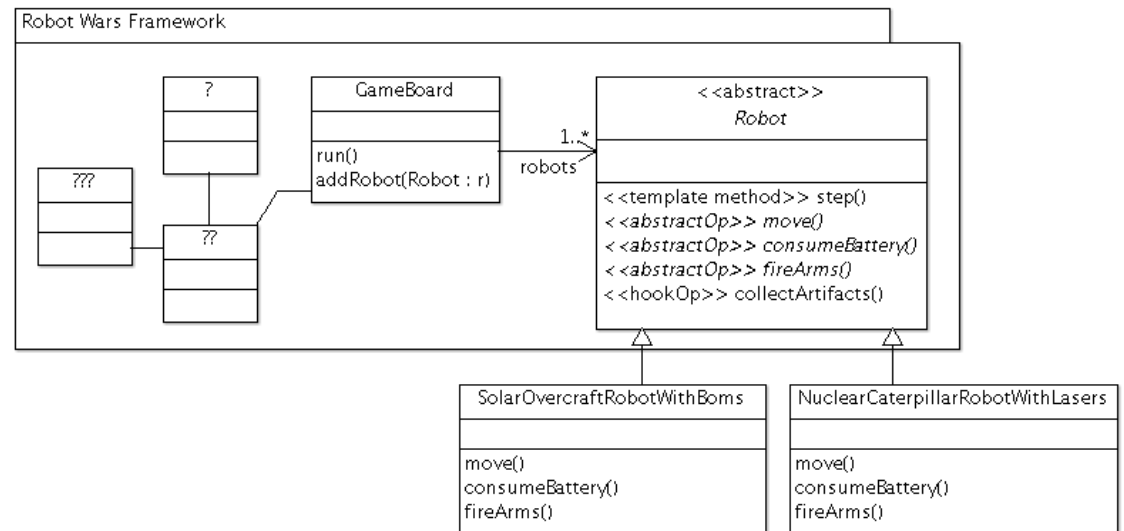


Para instanciar el framework:

- Subclasificamos Robot
- implementamos los métodos abstractos
- Usamos el estado heredado

En desarrollador del framework pone las reglas y sabe que esperar si las seguimos





```

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);
    }
}
  
```

SolarOvercraftRobotWithBoms
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");
}
```

NuclearCaterpillarRobotWithLasers
move() consumeBattery() fireArms()

```
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");
}
```

¿Como serán?  
SolarOvercraftRobotWithLasers  
SolarCaterpillarRobotWithLasers

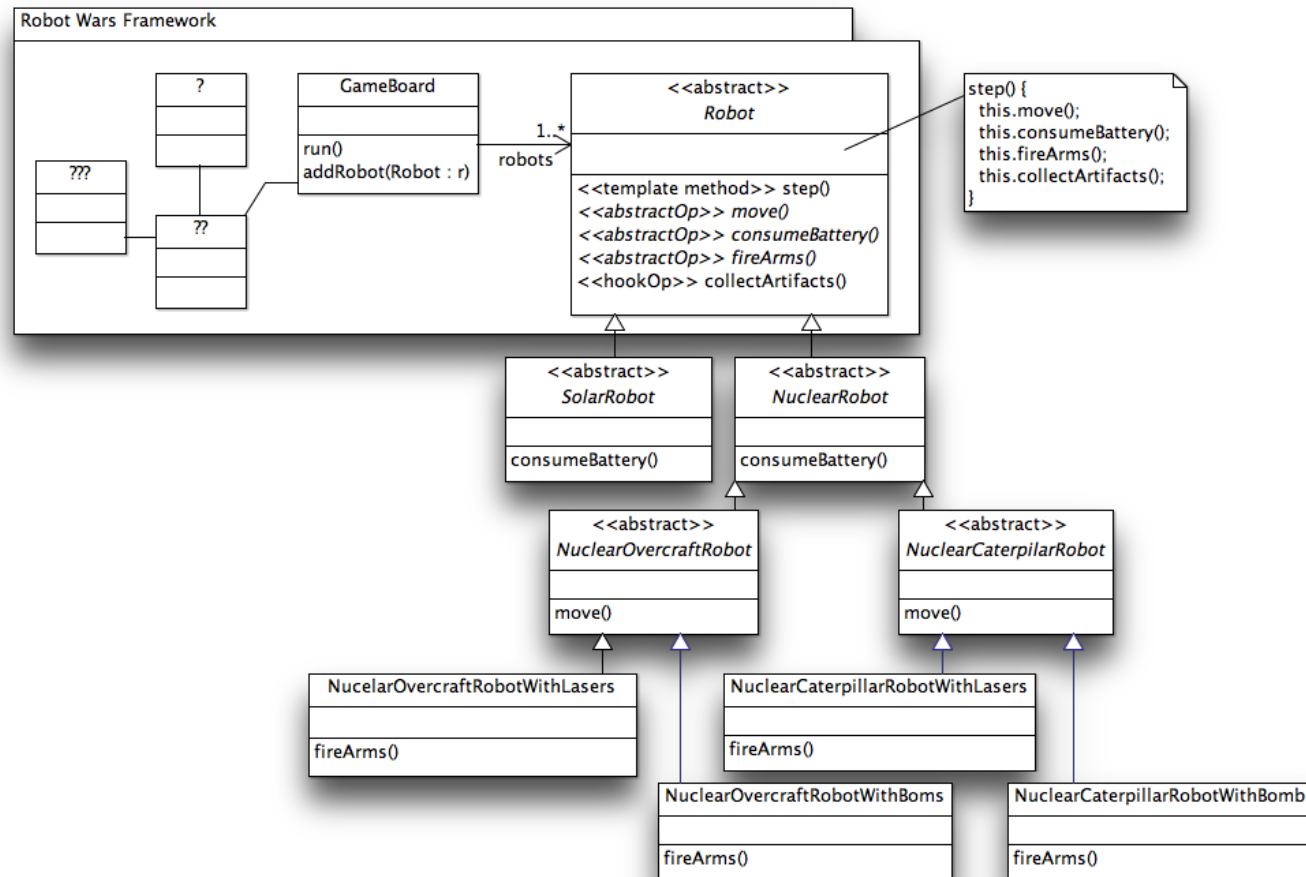
....

¿cómo lo resolvemos?

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

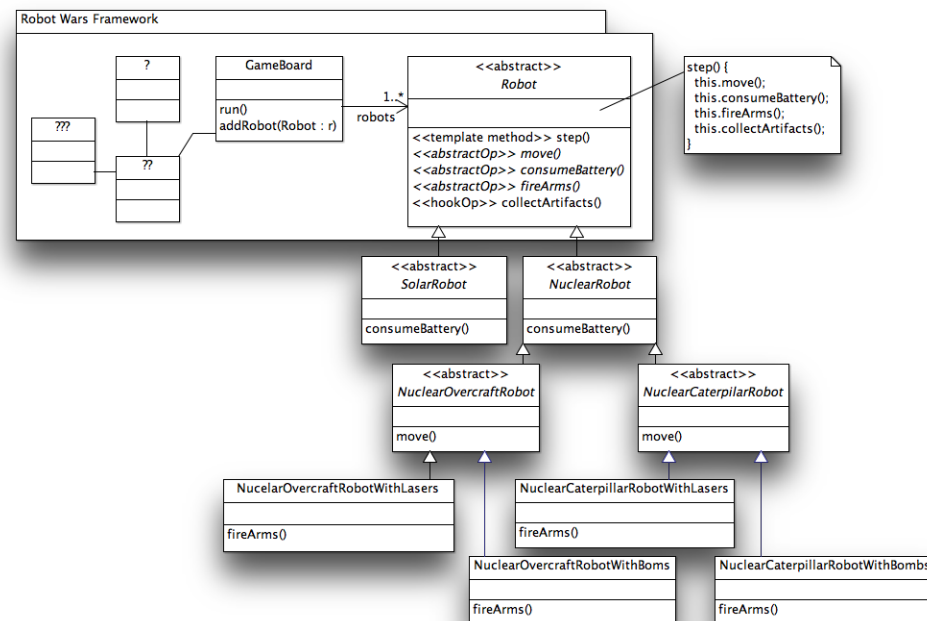


# Hot spots con herencia



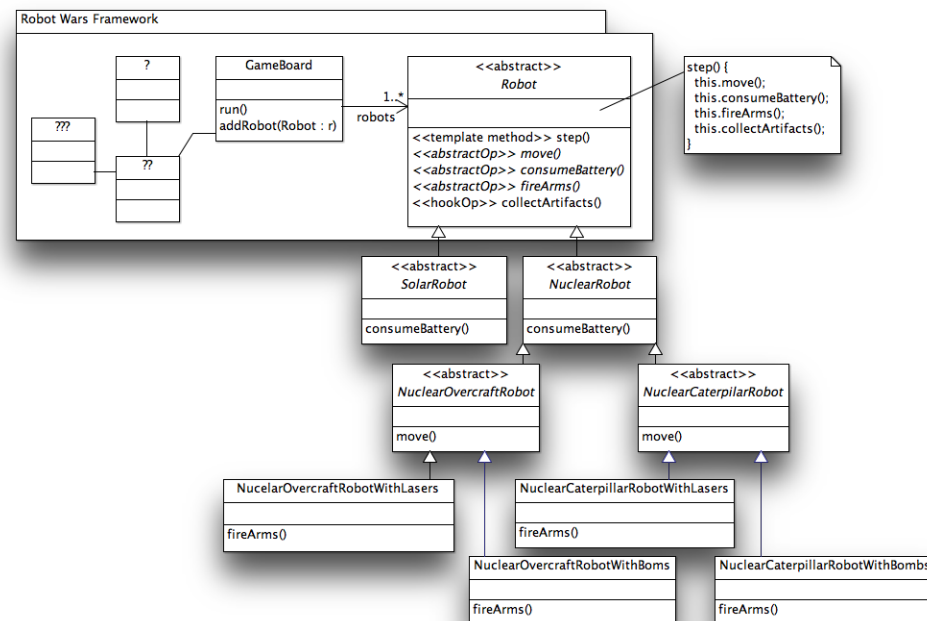
# Hot spots con herencia

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



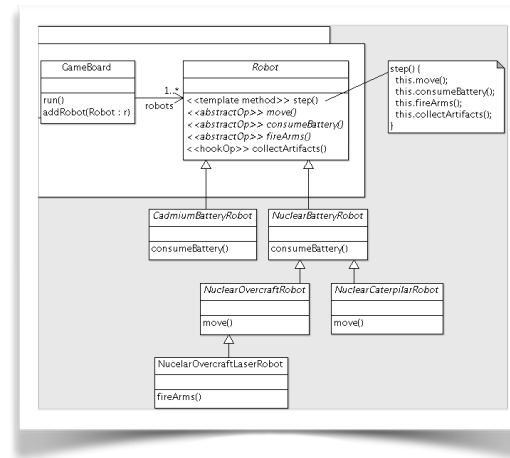
# Hot spots con herencia

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



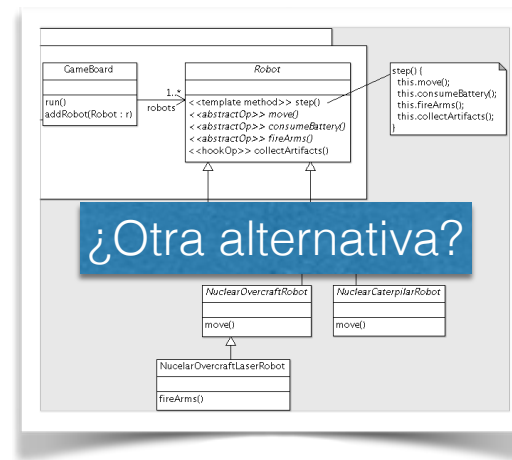
# Hot spots con herencia

- 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



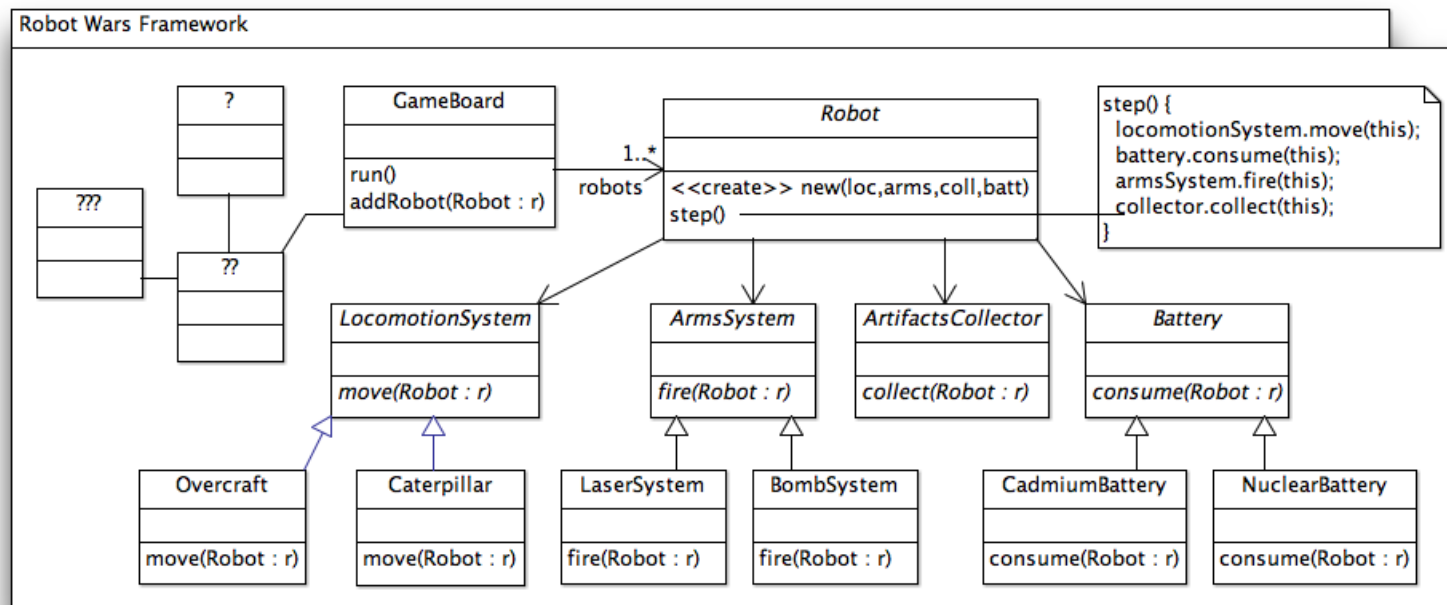
# Hot spots con herencia

- 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

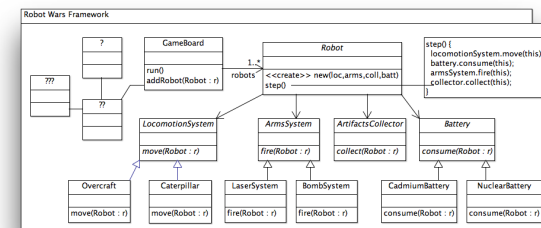


# Hotspots con composición

```
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.

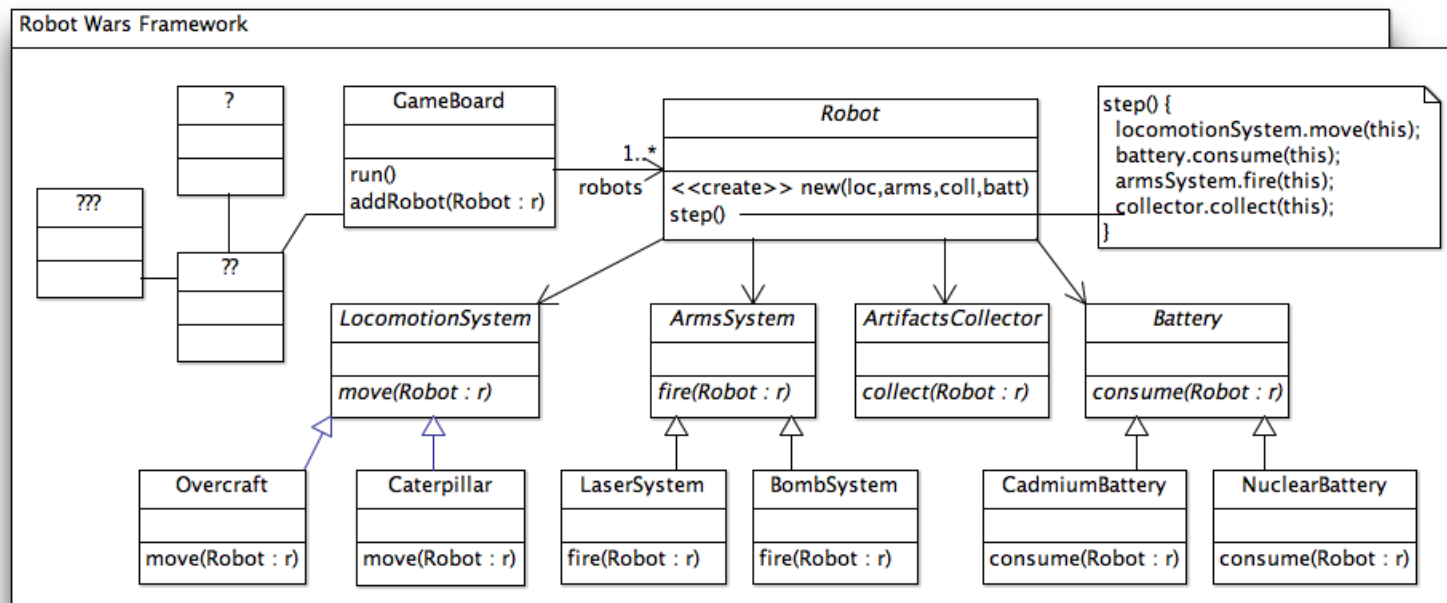


- Dependemos de las opciones provistas
- “Podríamos” sub-clasificar las componentes



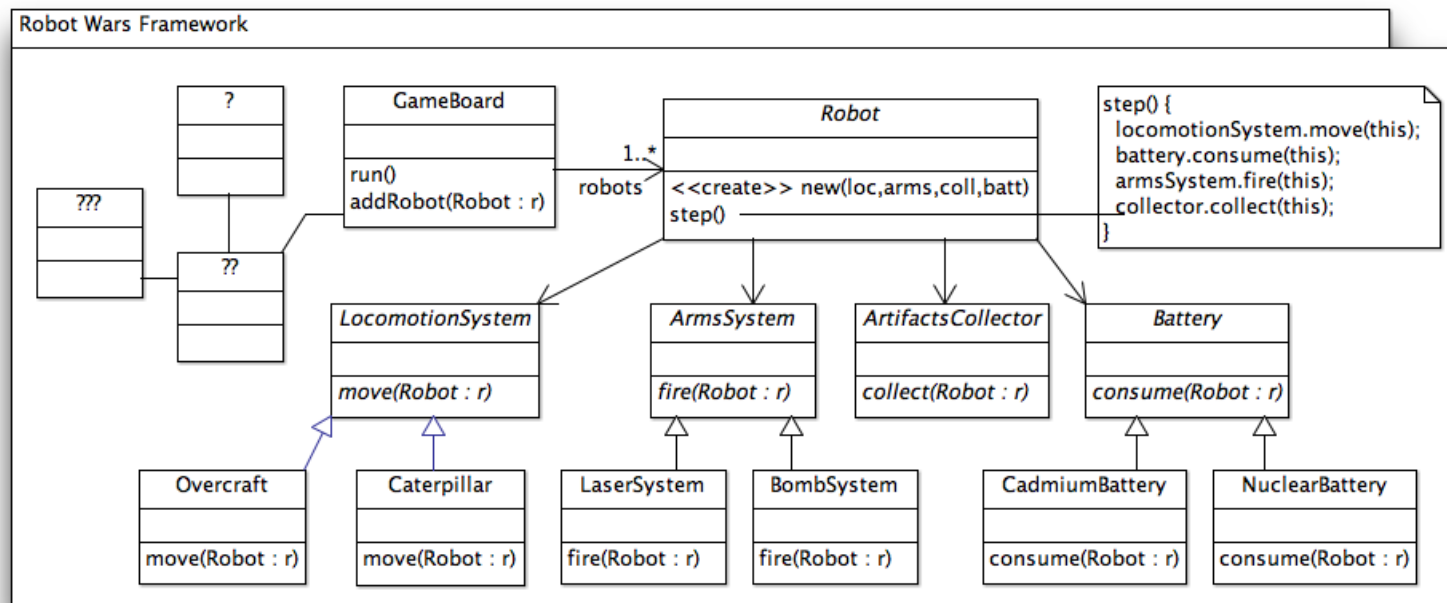
# Hotspots con composición

- ¿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?



# Plantillas y ganchos

- Hablamos de plantillas y ganchos como una forma de separar lo que es común de lo que varía
- El patrón template method implementa plantillas y ganchos con herencia (y mensajes a self)
- También podemos implementar plantillas y ganchos con composición (y delegación)
- En un framework, las plantillas suelen estar bajo control del desarrollador del framework (y las usa para implementar el frozen spot) que delega en el usuario la implementación/instanciación de los ganchos

# Hot-Spot-Driven Framework Development

Wolfgang Pree

Software Research Lab  
University of Constance  
D-78457 Constance, Germany  
Voice: +49-7531-88-44 33; Fax: +49-7531-88-35 77  
E-mail: pree@acm.org

**Abstract.** Most excellent object-oriented frameworks are still the product of a more or less chaotic development process, typically carried out in the realm of research-like settings. Overall, flexibility has to be injected into a framework in appropriate doses<sup>1</sup>. Framework adaptation takes place at points of predefined refinement that we call *hot spots*. As the quality of a framework depends directly on the appropriateness of hot spots, hot spot identification has to become an explicit activity in the framework development process. Means for documenting and communicating hot spots between domain experts and software engineers become crucial.

This contribution first discusses the few essential framework construction principles, that is, how to keep object-oriented architectures flexible for adaptations. We introduce hot spot cards as means to capture flexibility requirements, and illustrate how to apply them in combination with the essential framework construction principles. The presented heuristics form a hot-spot-driven framework design process which leads to a more systematic framework construction with fewer (re)design iterations.

**Key words:** framework hot spots, framework design, design patterns, frameworks, object-oriented design, object-oriented software development, software reusability

# 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.

