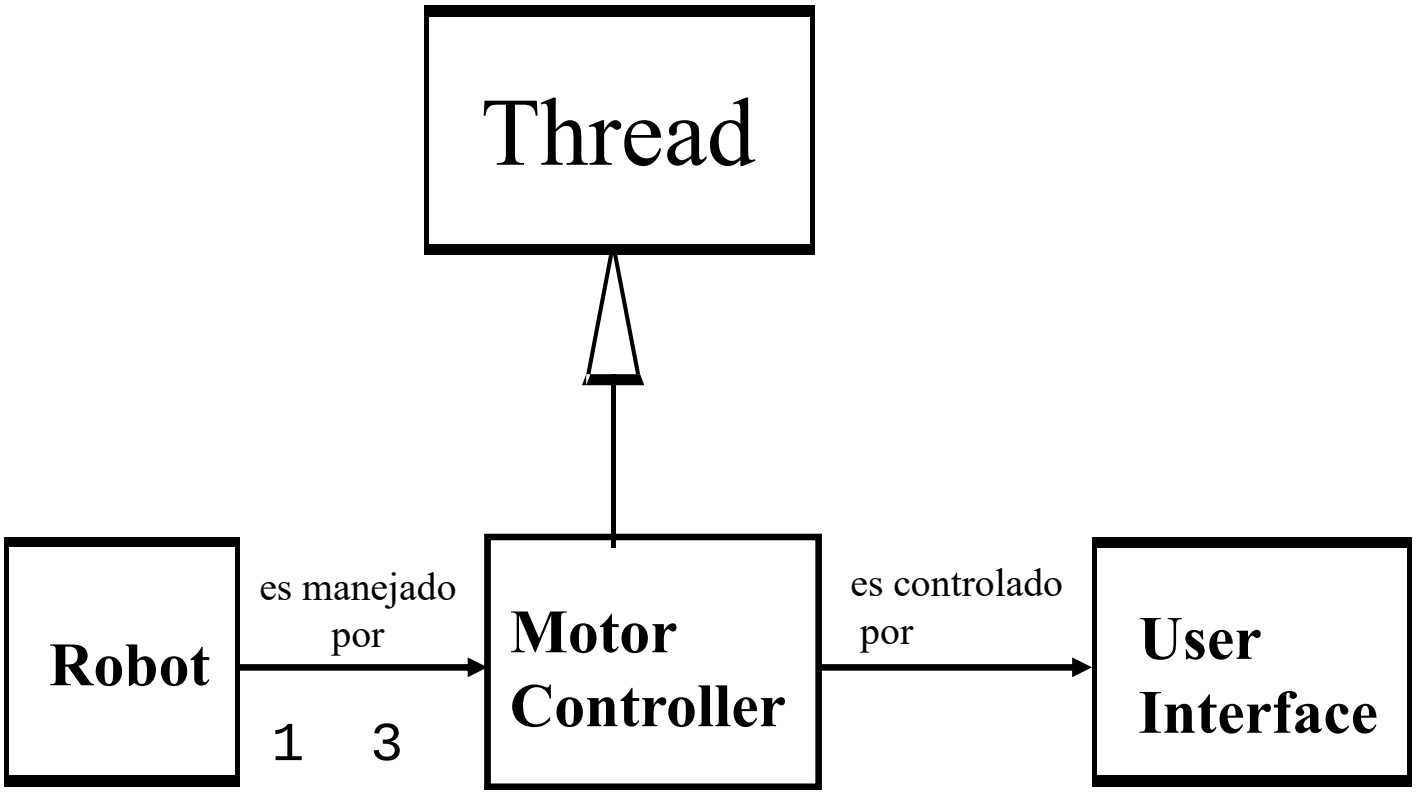
Ejemplo Robot

- Considere un robot que se puede mover en 3 dimensiones
- Un motor por separado controla el movimiento en cada dirección, y estos motores pueden ser operados simultáneamente para mover el robot a la posición indicada
- Los controladores de los motores son gobernados por una interfaz de usuario

Ejemplo Robot



Clases para Robot

```
public class UserInterface {
  // Permite que la siguiente posición del robot
  // sea obtenida de un operador
  public int newSetting (int dim) { ... }
  ...
}
public class Robot {
  // El Robot mismo.
  public void move(int dim, int pos) { ... }
  // Otros métodos.
}
```

MotorController deriva de Thread

```
public class MotorController extends Thread {

  private int dim;
  private UserInterface myInterface;
  private Robot myRobot;
  final int xPlane = 0;
  final int yPlane = 1;
  final int zPlane = 2;

//El parámetro dimension indica cual motor se controla
public MotorController(int dimension,
    UserInterface UI, Robot robo) {
      // constructor
    super();
    dim = dimension;
    myInterface = UI;
    myRobot = robo;
  }
```

MotorController deriva de Thread

```
public void run() {
  int position = 0; // posición inicial
  int setting;
  while(true) {
       // obtenga el offset y actualice la
       // posición
    setting = myInterface.newSetting(dim);
    position = position + setting;
       // muevase a la posición
       myRobot.move(dim, position);
  }
 }
}
```

método run sobrecargado

MotorController deriva de Thread

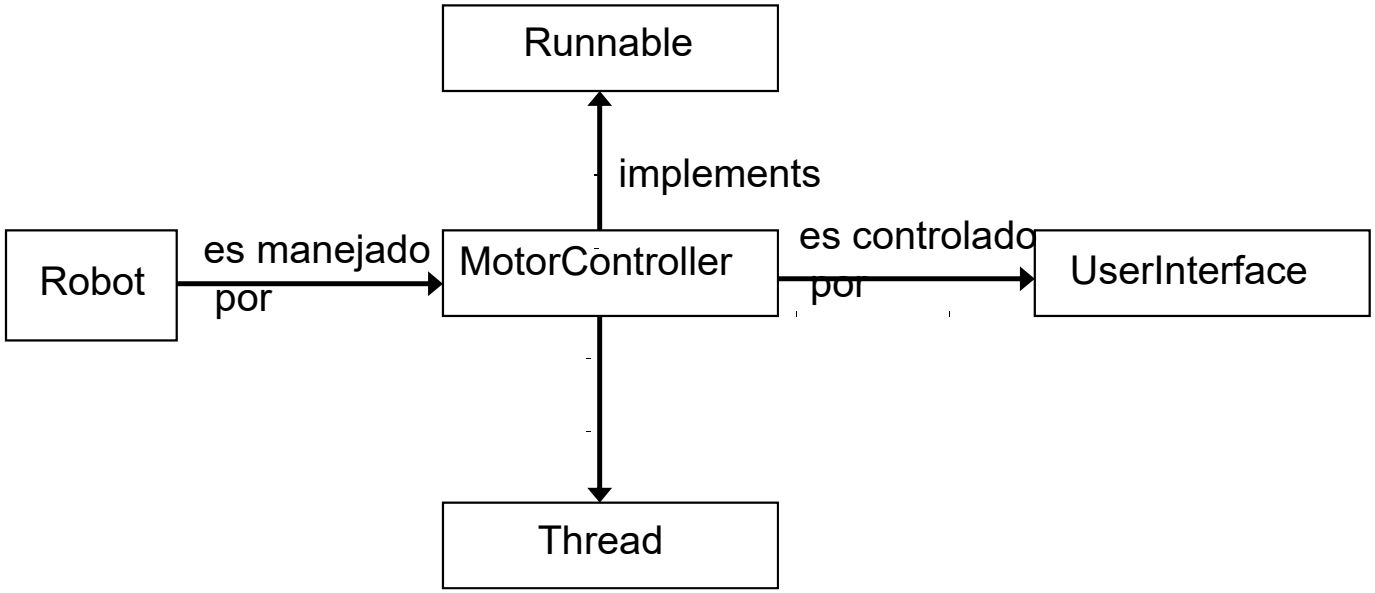
```
UserInterface UI = new UserInterface();
Robot robo= new Robot();

MotorController MC1 = new MotorController(
                          xPlane, UI, robo);

MotorController MC2 = new MotorController(
                          yPlane, UI, robo);

MotorController MC3 = new MotorController(
                          zPlane, UI, robo);

MC1.start();
MC2.start();
MC3.start();
```

creación de threads

MotorController deriva de Thread

- Cuando un thread es activado, se llama al método `run` y el thread se ejecuta
- Cuando el método `run` termina, el thread se considera terminado (Java denomina a esto estado muerto -dead state-)
- El thread permanece en este estado hasta que es eliminado por el garbage collector
- En nuestro ejemplo los threads no terminan

MotorController implementa la interfaz Runnable



MotorController implementa la interfaz Runnable

```
public class MotorController implements Runnable
{
   // Mismas variables que antes.
  public MotorController(int Dimension,
         UserInterface UI, Robot robo) {
    // El constructor es el mismo,
    // pero sin la llamada a super().
  }

  public void run() {
    // Método run idéntico.
  }
}
```

MotorController implementa Runnable

```
// Se construyen 3 controladores

UserInterface UI = new UserInterface();
Robot robo= new Robot();

MotorController MC1 = new MotorController(
                          xPlane, UI, robo);
MotorController MC2 = new MotorController(
                          yPlane, UI, robo);
MotorController MC3 = new MotorController(
                          zPlane, UI, robo);
```

Los threads no son creados todavía

MotorController implementa Runnable

```
Thread X = new Thread(MC1);
Thread Y = new Thread(MC2);
Thread Z = new Thread(MC2);

X.start();
Y.start();
Z.start();
```

los constructores pasaron un objeto (que implementa la interfaz `Runnable`) cuando los threads fueron creados

los threads comienzan