Polymorphism and Interfaces

COMP1400 - Week 11

this

The keyword `this` is can be used by an object as a reference to itself.

It is usually used when telling other objects about itself:

```
room.enter(this);
```

this

on Player:

```
public void moveTo(Room r) {
    r.enter(this);
}
```

on Room:

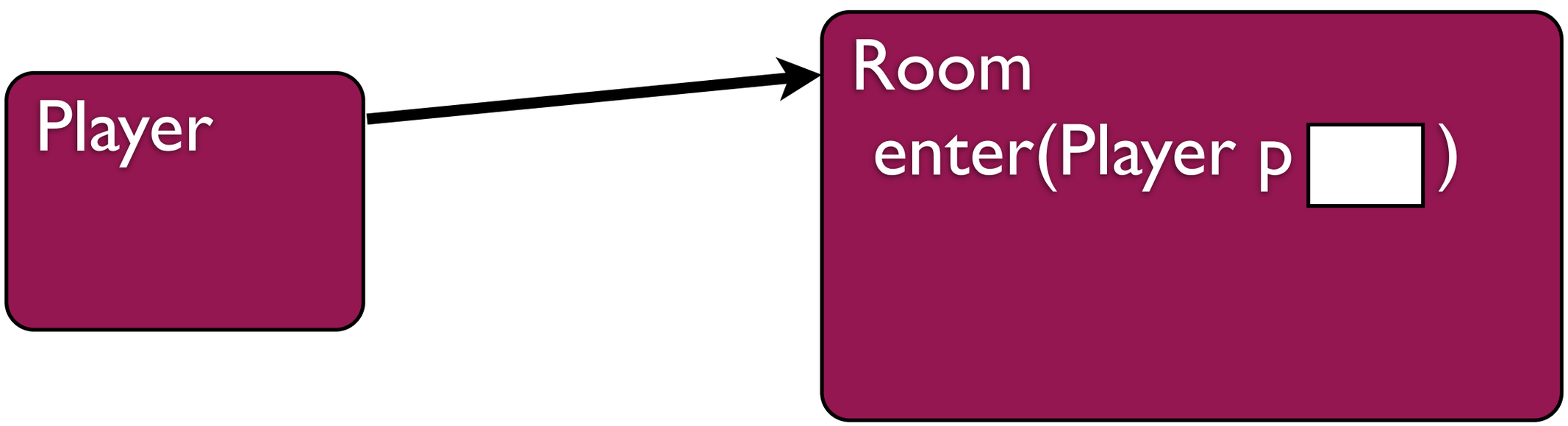
```
public void enter(Player p) {
    // they found an arrow!
    p.addArrows(1);
}
```

Removing ambiguity

```
public class RobotGame {

   private int robots;  // BAD
                        // STYLE

   public RobotGame(int robots) {

      // field          parameter
      this.robots = robots;
   }
}
```

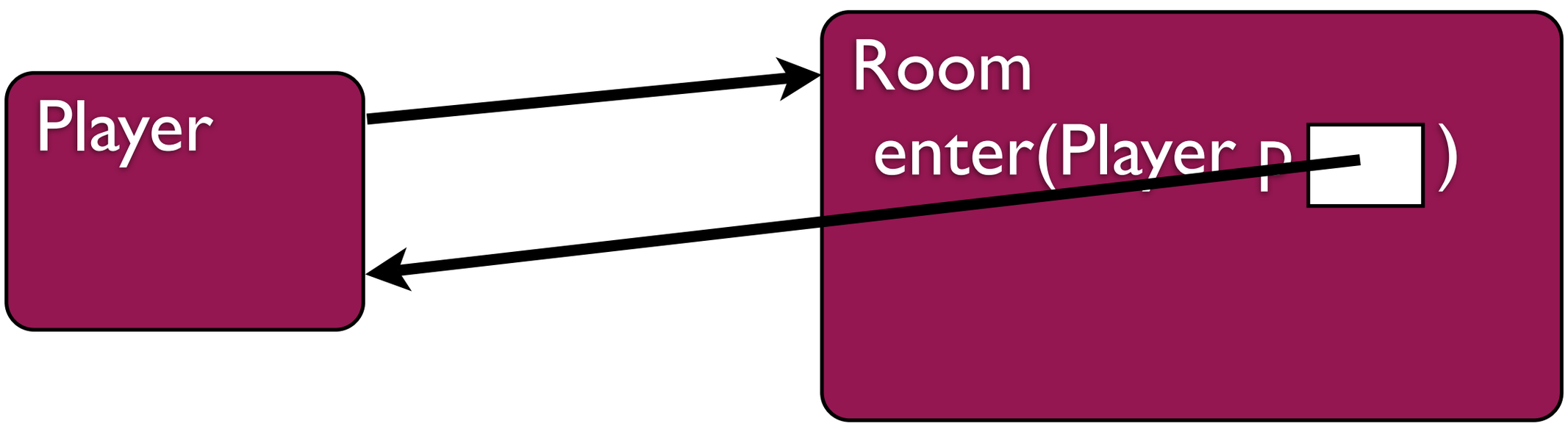
this

```
public void moveTo(Room r) {
   r.enter(this);
}
```



this

```
public void moveTo(Room r) {
   r.enter(this);
}
```



Polymorphism

In the real world a particular object may be thought to belong to a number of different categories (types) in different contexts

Malcolm is

- a person
- a man
- a lecturer
- a mammal
- a musician
- a 70kg mass

Polymorphism

Such an object is polymorphic. It can have different types in different circumstances.

A polymorphic object implements a number of different interfaces.

Each interface defines an expected set of methods by which it can be used.

Interfaces

A lecturer can be asked to:

- teach
- mark

A musician can be asked to:

- play music

A 70kg mass can be asked to:

- accelerate

Interfaces

The object (Malcolm) must implement all of these interfaces.

Different objects may implement interfaces differently.

Eg: Malcolm implements "play music" using a ukulele.

Another musician might use a trombone or a glockenspiel.

Interfaces in Java

In Java we define interfaces like empty classes:

```
public interface Hazard {
    // interface definition.

    public String getWarning();

    public boolean
        activate(Player player);

}
```

Interfaces in Java

'interface' keyword

interface name

In Java we define interfaces like empty classes:

```
public interface Hazard {
    // interface definition.

    public String getWarning();

    public boolean
        activate(Player player);
}
```

method signatures

no method bodies

semicolon

Interfaces in Java

Interfaces contain no code or data,
only method signatures.

They do nothing on their own.

They merely describe interfaces for other classes.

Interfaces in Java

Objects must list the interfaces they implement

```
public class Bats
        implements Hazard {
    // class definition ...
}
```

Interfaces in Java

'implements' keyword

interface name

Objects m terfaces they

```
public class Bats
        implements Hazard {
    // class definition ...
}
```

Interfaces in Java

A class which implements an interface must provide methods that match those in the interface description.

Multiple classes may implement the same interface in different ways.

```
public class Bats
        implements Hazard {

    public String getWarning() {

        return "You hear squeaking.";

    }

    public boolean
         activate(Player player) {

        // ... move the player ...

        return false;
    }
}
```

```
public class Pit
        implements Hazard {

    public String getWarning() {

        return "You fell a draft.";

    }

    public boolean
         activate(Player player) {

        // ... kill the player ...

        return true;
    }
}
```

Using Interfaces

An object which implements an interface may be treated as an instance of that type:

```
Bats b = new Bats();
Hazard h = b;

String warn = h.getWarning();

boolean gameOver =
        h.activate(player);
```

```
private ArrayList<Hazard>
                              myHazards;

public void addHazard(Hazard h)  {

    myHazards.add(h);

}

public void activateHazards(
                        Player p)  {

    for (Hazard h : myHazards)  {
        h.activate(p);
    }
}
```

Interface example

```
Room room = new Room();

Bats bats = new Bats();

room.addHazard(bats);

Room room2 = new Room();

Pit pit = new Pit();

room2.addHazard(pit);
```

Using Interfaces

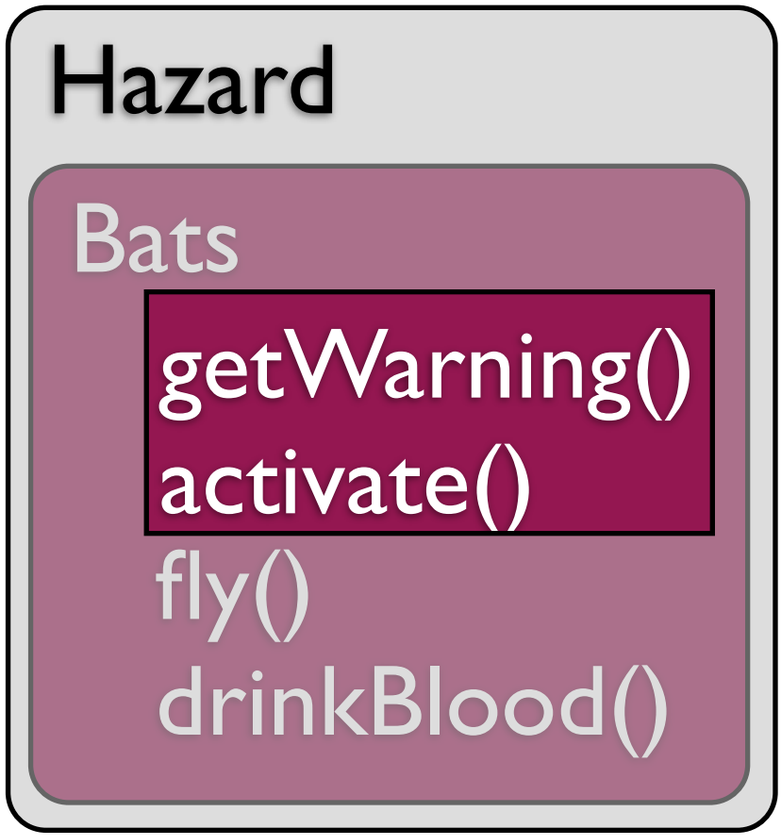
Note that while all Bats are Hazards, not all Hazards are Bats. So the following is wrong:

```
Hazard h = new Pit();

Bats b = h;   // WRONG!

h = new Bats();

b = h;   // WRONG!
```

Interfaces as Masks

It is helpful to visualise interfaces as masks which only reveal parts of a class:

```
Bats b = new Bats();
Hazard h = b;

String warn =
    h.getWarning();


h.fly(); // ERROR
```



Design

Interfaces are a useful tool for abstraction.

Often we have several different objects which are functionally similar at the abstract level but differ in implementation detail.

Interfaces allow us to ignore the details when they are irrelevant.

Interfaces in JCL

The Java Class Library includes a number of interfaces:

Eg: The **`List`** interface abstracts the idea of a sorted list.

The **`ArrayList`** class implements this interface. So does **`LinkedList`**.

http://docs.oracle.com/javase/1.4.2/docs/api/

Lists

```
public Room {
   private List<Hazard> myHazards;
   public Room() {
      myHazards =
        new ArrayList<Hazard>();
   }
}
```

Lists

```
public Room {
   private List<Hazard> myHazards;
   public Room() {
      myHazards =
        new LinkedList<Hazard>();
   }
}
```

Comparable

One important interface is Comparable:

http://docs.oracle.com/javase/1.4.2/docs/api/java/lang/Comparable.html

It describes the standard method for comparing objects:

```
public int compareTo(Object o);
```