Example: PacMan and SuperPacMan

```java
public class PacMan {
    private int x;
    private int y;

    public PacMan(int x, int y) {}
    public void eatDot() {}
    public void move() {}
    public void turnLeft() {}
}

public class SuperPacMan extends PacMan {
    public void turnRight() {
        turnLeft();
        turnLeft();
        turnLeft();
    }

    public class SuperPacMan extends PacMan {
        public void turnRight() {
            turnLeft();
            turnLeft();
            turnLeft();
        }
    }
}
```

Inheritance

One of the key concepts of OOP

- A hierarchical relationship among classes
- Establishes a superclass/subclass relationship
- Establishes "is a" relationships
  - e.g. a SuperPacMan "is a" PacMan

Benefits:

- Reusability of code
  - Put code in one class, use it in all the subclasses
  - Revisions only need to be done in 1 place
- Polymorphic code (works on "many forms")
  - Write general purpose code designed for a supertype that works for all subtypes

The “extends” keyword

Inheritance is established via the “extends” keyword

```java
public class PacMan {
}

public class SuperPacMan extends PacMan {
}
```

Now we say

- SuperPacMan inherits from PacMan
- A SuperPacMan is a "is a" PacMan
- SuperPacMan is a subclass/subtype of PacMan
- PacMan is the superclass/supertype of SuperPacMan
What can you inherit?

- Visibility modifiers determine which class members are accessible and which do not

- Members (variables and methods) declared with `public` visibility are accessible, and those with `private` visibility are not

- Problem: How to make class instance variables visible only to its subclasses?
  - Solution: Java provides a third visibility modifier that helps in inheritance situations: `protected`

Inheritance Tree

- Java has *single inheritance*; each node has one parent
- Except for `Object` which has no parent

The Object Class

All classes inherit from the `Object` class

- The `Object` class is the root of the class hierarchy
- When we create a new class, "extends Object" is implied/implicit

```java
class Car {
}
class Car extends Object {
}
```

The `Object` class has several methods which all objects inherit, most notably: `toString()` and `equals()`

- Once we inherit these, we can also override the behavior i.e. make it conform to what the object of subclass will do when this method is called.

Need for `equals()`: Comparison of Strings

- If the `==` operator is used for Strings
  - Java compares the addresses where the String objects are stored, not the letters in the String
  - For example:
    ```java
    String a = "hi";
    String b = "hi";
    result = a == b
    result: false
    ```

- Use the `String` class' `equals` method to compare two Strings for equality
  ```java
  a.equals(b)
  ```
The equals() Method

- By default, compares heap addresses
- By convention, it is overridden to match the developer’s notion of equality

```java
public class Person {
    private int social;  // social security #
    private String name;

    public int getSocial() { return social; }  
    public boolean equals(Person p) {
        return this.social == p.getSocial();
    }
}
```

The toString() Method

- By default, it returns a String containing an object’s heap address
- By convention, it is overridden to describe the object’s state
- Most common usage: debugging

```java
public class Car {
    private int miles;private String model;

    public Car(String model) {
        this.model = model;
        miles = 0;
    }

    public String toString() {
        return "model: " + model + ", miles: " + miles;
    }
}
```

Constructors and Inheritance

SuperPacMan b = new SuperPacMan(0,0);

- When an object is created, its constructor is called
- But first, a constructor from its highest ancestor (Object) is called, then the next highest (PacMan), then its own (SuperPacMan)
- The default behavior is such that the default (no-argument constructor) is executed

A constructor can explicitly call its parent’s (its superclass’) constructor by making a call to super(arguments)

```java
public class SuperPacMan extends PacMan{  
    public SuperPacMan (int x, int y){  
        super(x, y);  
        ...
    }
}
```

Polymorphism

Polymorphism means many (poly) shapes (morph)

In Java, polymorphism refers to the fact that you can have multiple methods with the same name in the same class

There are two kinds of polymorphism:

- Overloading
  - Two or more methods with different signatures
- Overriding
  - A method in a subclass to “override” a method in the superclass that has the same signature

We’ve already seen Overloading scenario with Constructors
E.g. public BoeBotControl() {..}
    public BoeBotControl(PWM l, PWM r) {...}
Method Overloading

Method overloading occurs when

- A class has two or more methods with the same name but different signatures
  - Different signature -> the number, order, or types of their parameters differ

```java
// the foo method is overloaded
public void foo() {...}
public void foo(int x) {...}
public void foo(double x){...}
public void foo(int x, double y){...}
```

Overriding

Overriding occurs if

- There are two or more methods with the same name and the same signature in an inheritance chain
- For example, the Object class has a toString() method
  - It can be overridden in a subclass simply by creating a method with the same signature
    ```java
    public String toString() {...}
    ```
  - Java picks the “lowest” method in the inheritance chain possible

Type Rules

- A reference variable of type `t` may hold a value of its own type or any subtype (but not of a supertype).
- Given the following variable declaration:
  ```java
  PacMan b;
  ```
  Which of the following assignments are valid (compile)?
  ```java
  b = new PacMan(0, 0);
b = new String();
b = new SuperPacMan(0, 0);
b = new Object();
  ```
  How about these?
  ```java
  SuperPacMan bb;
  bb = new PacMan(0, 0);
  bb = new String();
  bb = new SuperPacMan(0, 0);
  bb = new Object();
  ```

The “instanceof” Operator

```java
PacMan b = new PacMan(0, 0);
b instanceof PacMan
Answer: true
b instanceof Object
Answer: true
b instanceof SuperPacMan
Answer: false
```
The “instanceof” Operator

```java
SuperPacMan bb = new SuperPacMan(0, 0);
bb instanceof SuperPacMan
Answer: true

bb instanceof Object
Answer: true

bb instanceof PacMan
Answer: true

PacMan bbb = new SuperPacMan(0, 0);  // a variable can store a subtype

bbb instanceof SuperPacMan(0, 0)
Answer: true

bbb instanceof PacMan
Answer: true
```