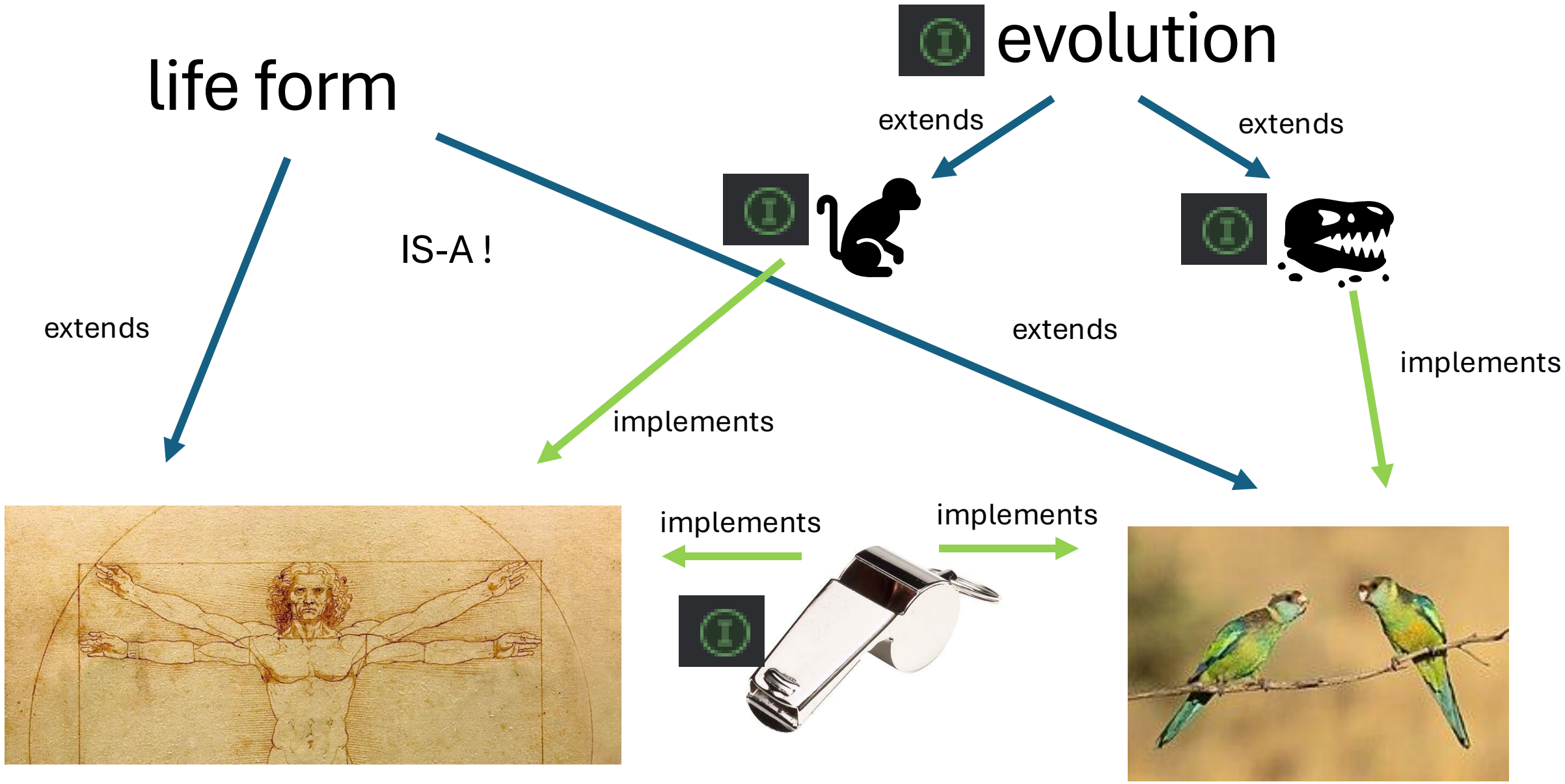
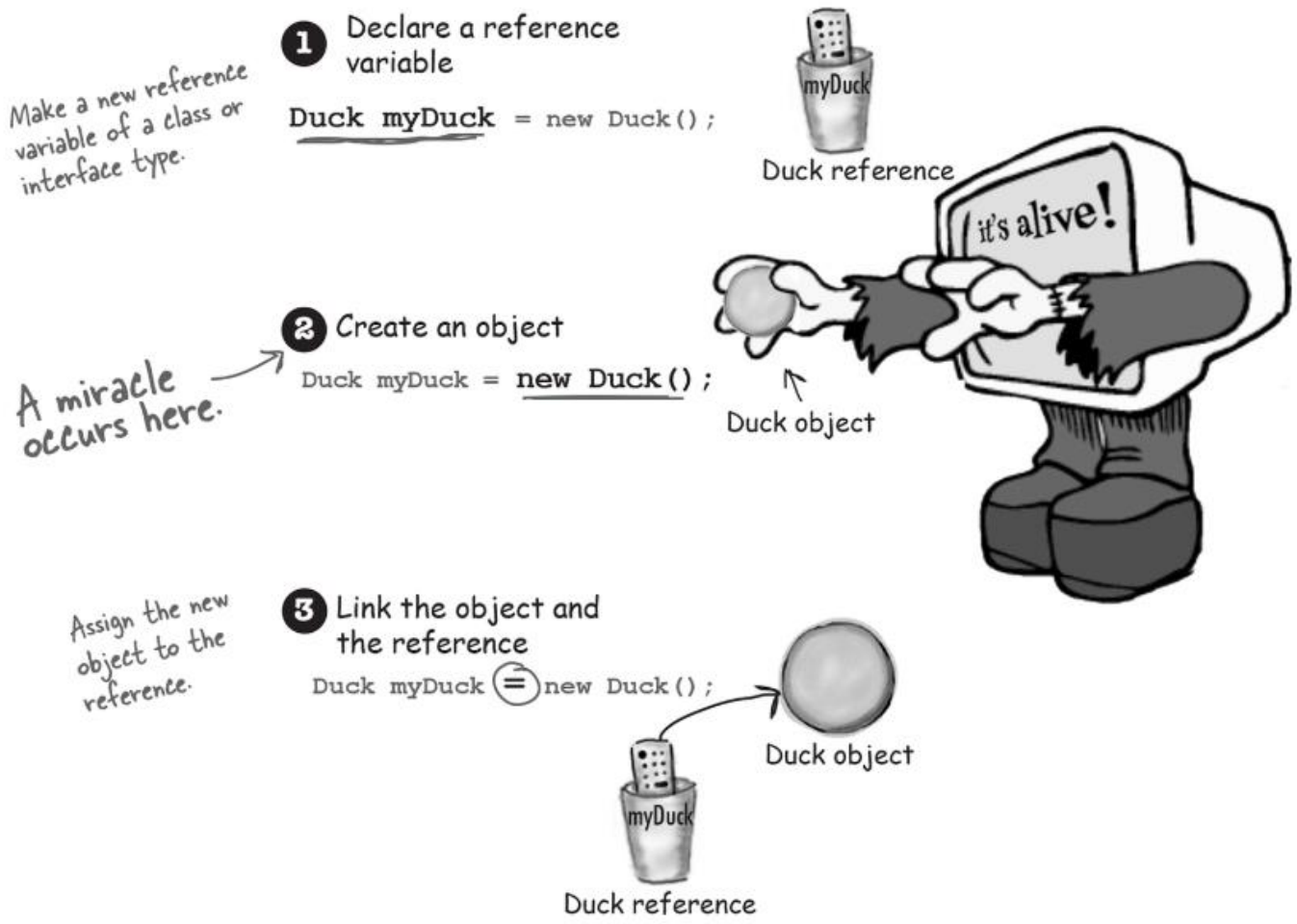
Konstruktører

Uke10Xtra

Repetisjon: Polimorfi&Interface


life form
evolution
extends
extends
IS-A !
extends
extends
implements
implements
implements
implements

Konstruktør



Konstruktør

Are we calling a method named Duck()? Because it sure *looks* like it.

```
Duck myDuck = new Duck();
```

It looks like we're calling a method named Duck(), because of the parentheses.

No.

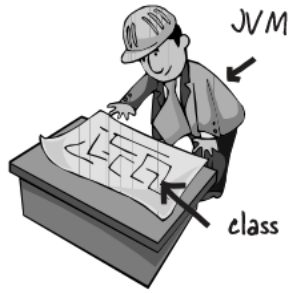
We're calling the Duck *constructor*.

```
public class Duck {

    public Duck() {
        System.out.println("Quack");
    }
}
```

Constructor code.

A class is not an object (but it's used to construct them)



Konstruktør

```
public class Duck2 {
  int size;

  public Duck2() {
    // supply default size
    size = 27;
  }

  public Duck2(int duckSize) {
    // use duckSize parameter
    size = duckSize;
  }
}
```

If you have more than one constructor in a class, it means you have overloaded constructors.

To make a Duck when you know the size:

```
Duck2 d = new Duck2(15);
```

To make a Duck when you do not know the size:

```
Duck2 d2 = new Duck2();
```

Konstruktør

If you have more than one constructor in a class, it means you have overloaded constructors.

Five different constructors means five different ways to make a new mushroom.

```
public class Mushroom {

  public Mushroom(int size) { }

  public Mushroom( ) { }

  public Mushroom(boolean isMagic) { }

  public Mushroom(boolean isMagic, int size) { }

  public Mushroom(int size, boolean isMagic) { }

}
```

When you know the size, but you don't know if it's magic

When you don't know anything

When you know if it's magic or not, but don't know the size

These two have the same args, but in a different order, so it's OK*

When you know whether or not it's magic, AND you know the size as well

*If the arguments were the same type, how would the compiler know they were two different things?

Konstruktør

```
public class Duck extends Animal {
  int size;

  public Duck(int newSize) {
    super();
    size = newSize;
  }
}
```

you just call super()

If you don't provide a constructor

The compiler puts one

Konstruktør

```
import java.awt.Color;

class Mini extends Car {
  private Color color;

  public Mini() {
    this(Color.RED);
  }

  public Mini(Color c) {
    super("Mini");
    color = c;
    // more initialization
  }

  publi      t size) {
    is(  lor.R  );
    s  er(siz
  }
}
```

The no-arg constructor supplies a default Color and calls the overloaded Real Constructor (the one that calls super()).

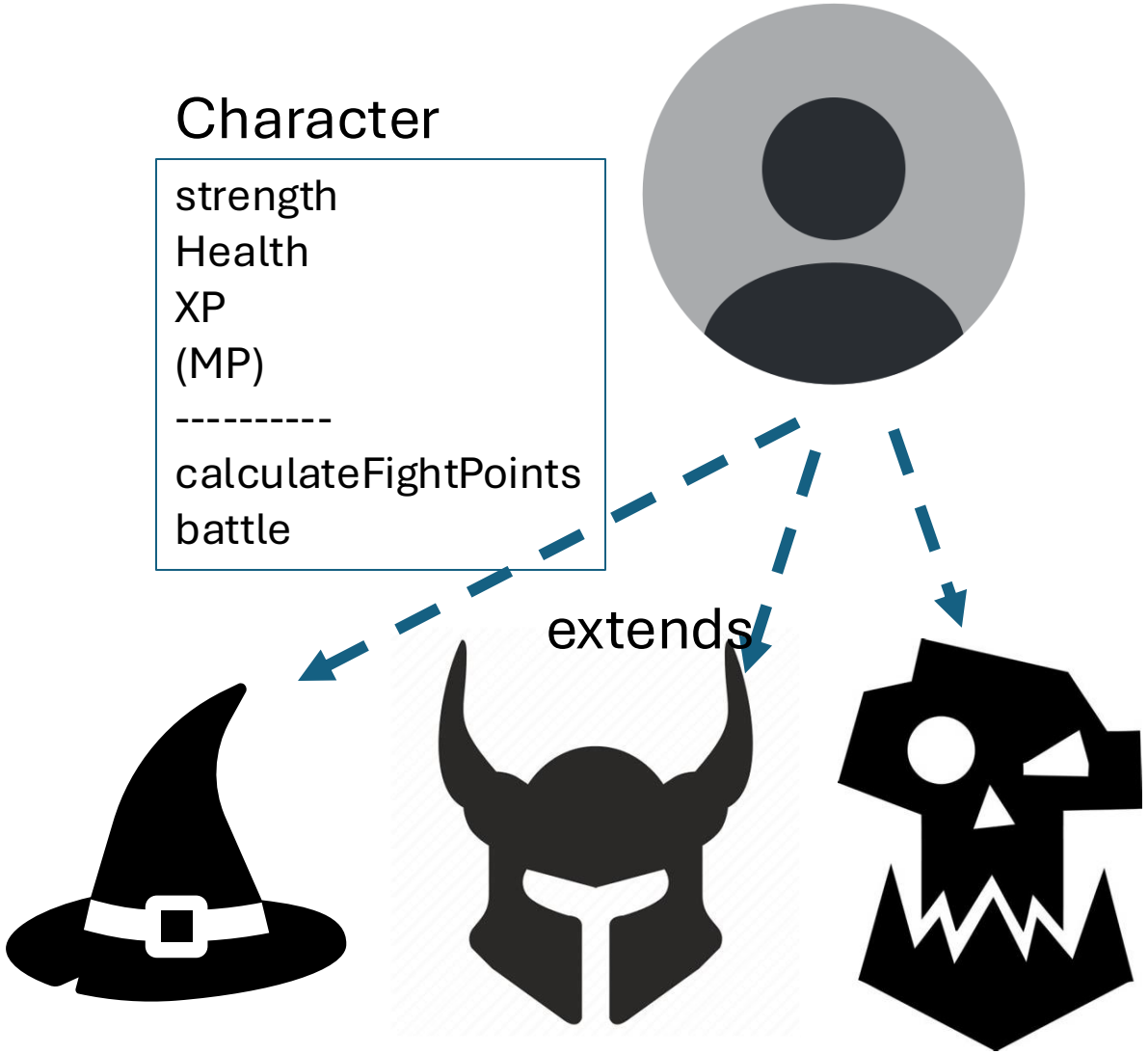
This is The Real Constructor that does The Real Work of initializing the object (including the call to super()).

Won't work!! Can't have super() and this() in the same constructor, because they each must be the first statement!

```
File Edit Window Help Drive
javac Mini.java

Mini.java:16: call to super must
be first statement in constructor
      super();
            ^
```

»coding-eksempel» mini-RPG



That ´s all folks

...

Feel free to use my RPG code for playing around..

weaponize the characters