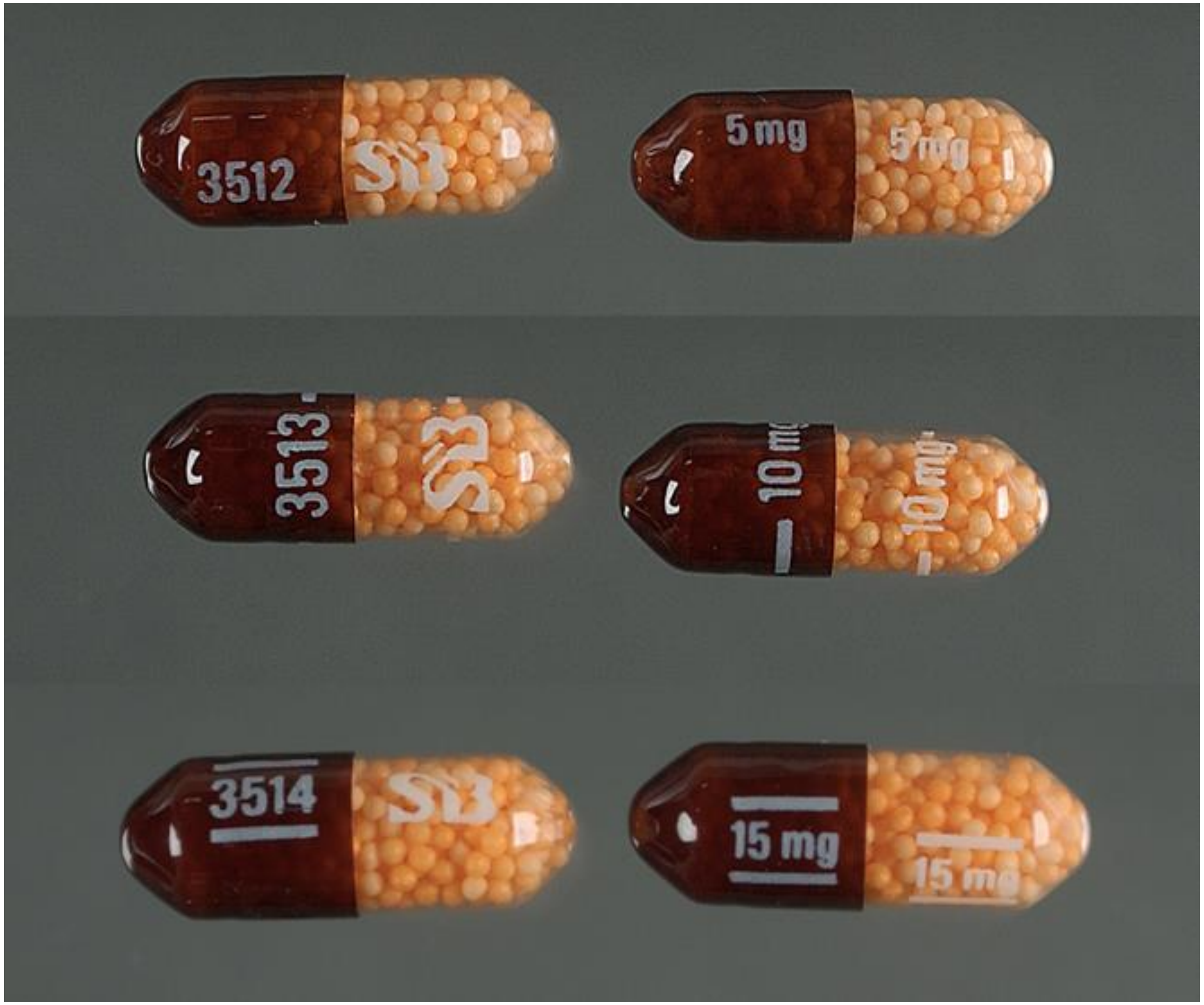
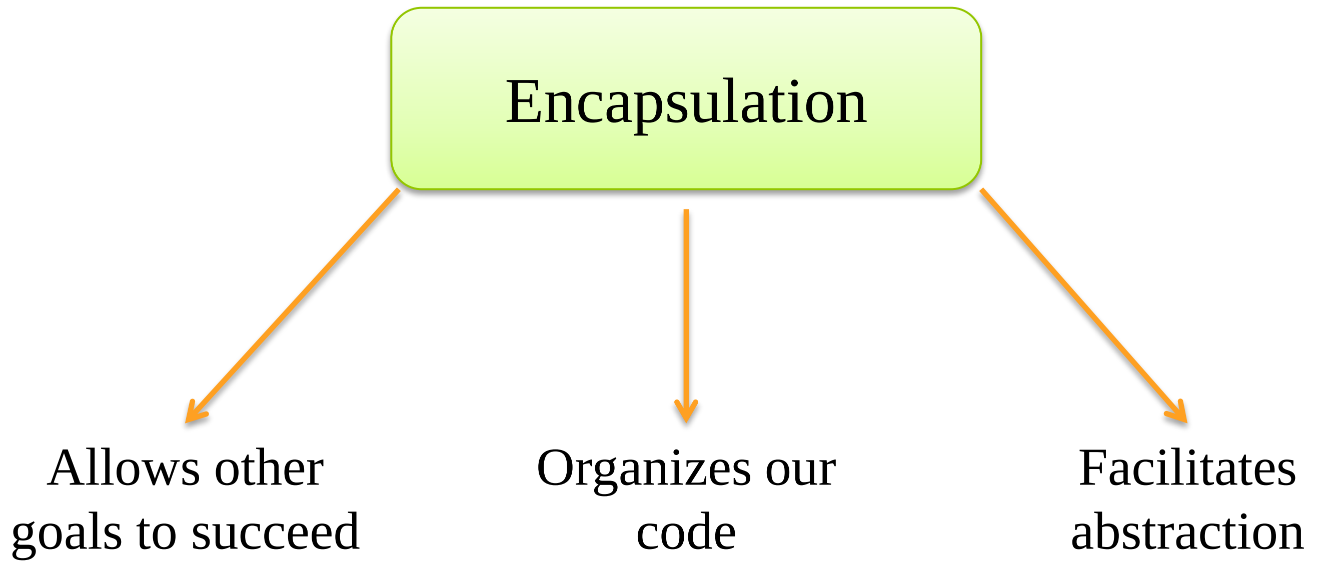
Goals of Class Use

Encapsulation
Code Reuse
Dividing the Problem
Information Hiding
Readability and Expressiveness

Encapsulation



http://commons.wikimedia.org/wiki/File:Dexedrine_doj2.jpeg


Encapsulation
Allows other goals to succeed
Organizes our code
Facilitates abstraction

Code Reuse

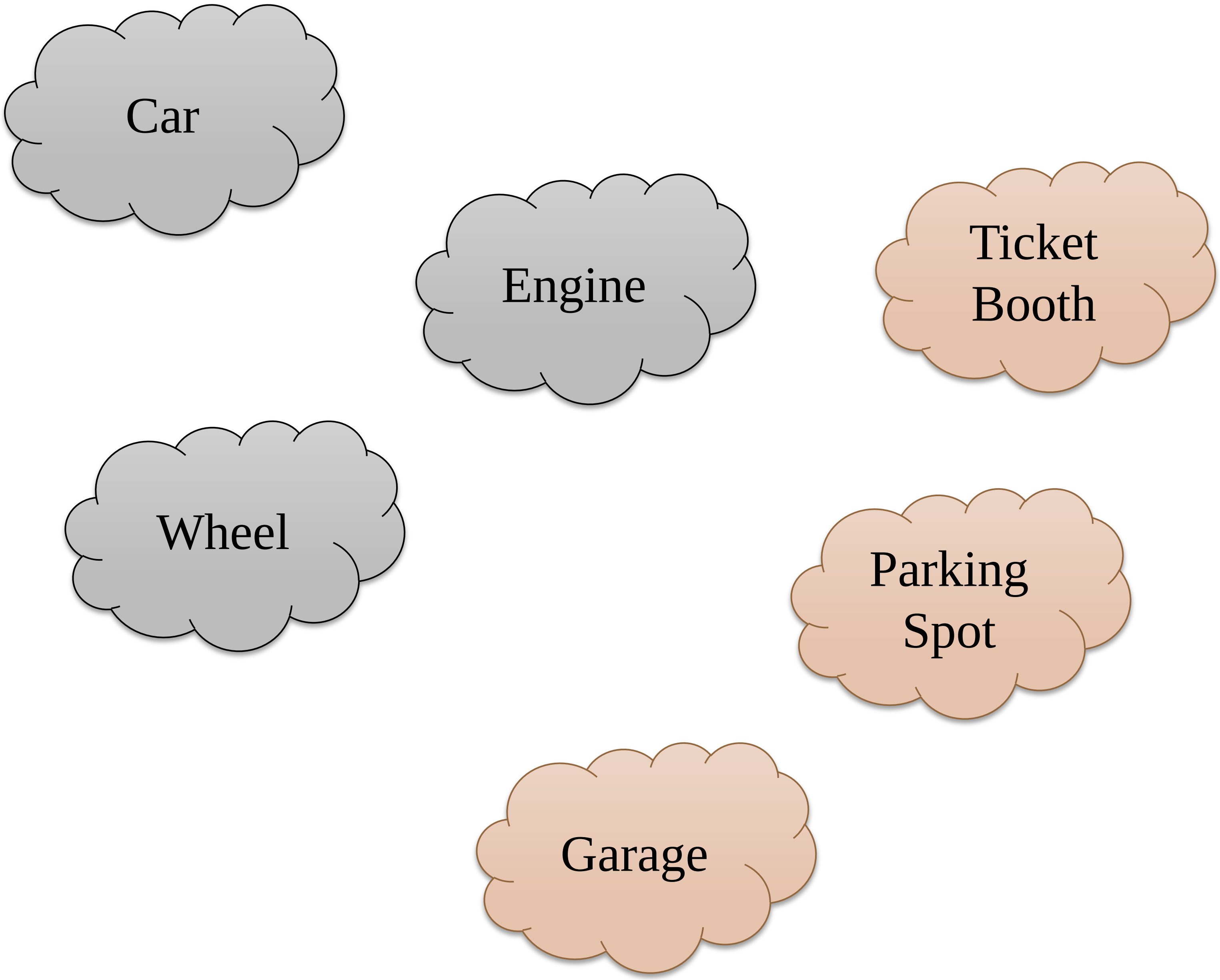
http://commons.wikimedia.org/wiki/File:USB_TypeA_Plug.JPG

To reuse a class elsewhere, it has to be independent from the rest of the program.

To reuse a class elsewhere, it has to be independent from the rest of the program.

Reuse can also come in the form of inheritance, discussed later in the course.

Dividing the Problem

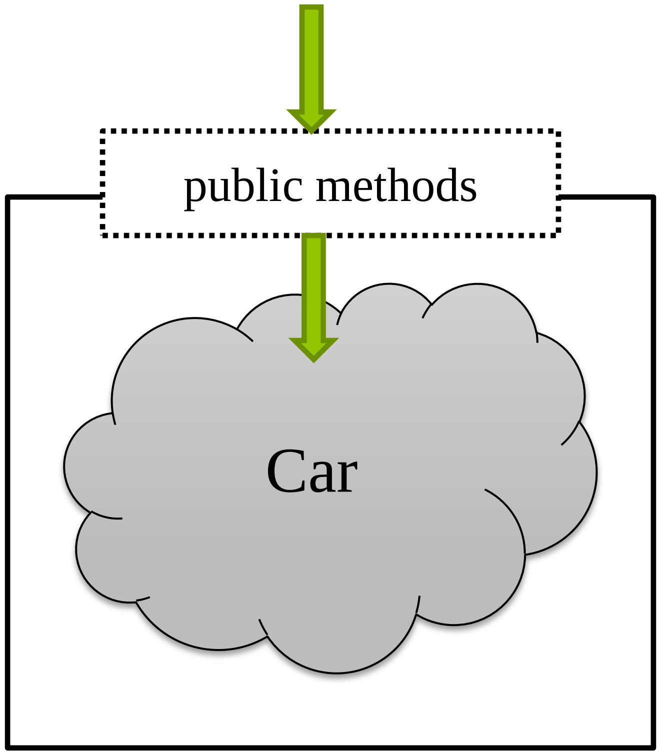

Car
Engine
Ticket Booth
Wheel
Parking Spot
Garage

Let well encapsulated classes be the
method of dividing complex problems.

Let well encapsulated classes be the
method of dividing complex problems.

Side effect: reusable code!

Information Hiding



Keep a consistent interface

http://www.i-am-geek.net/the-last-of-us/

Don't let other objects have access to your private parts

Access modifier:

permission setting for our attributes and methods so that they will be visible/modifiable/usable from some places in our code but not from other places

In Java:

`public` (everywhere)
`<blank>` (same package/folder)
`protected` (this class and its subclasses)
`private` (only this class)

Access modifier:

permission setting for our attributes and methods so that they will be visible/modifiable/usable from some places in our code but not

lces

Most methods we write in the class

`public` (everywhere)

`<blank>` (same package/folder)

`protected` (this class and its subclasses)

`private` (only this class)

Access modifier:

permission setting for our attributes and methods so that they will be visible/modifiable/usable from some places in our code but not from other places

Same as public when all classes in same folder

`public` (everywhere)

`<blank>` (same package/folder)

`protected` (this class and its subclasses)

`private` (only this class)

Access modifier:

permission setting for our attributes and methods so that they will be visible/modifiable/usable from some places in our code but not from other places

More on subclasses later on

[illegible] (everywhere)

[illegible] ame package/folder)

`protected` (this class and its subclasses)

`private` (only this class)

Access modifier:

permission setting for our attributes and methods so that they will be visible/modifiable/usable from some places in our code but not from other places

`public` (everywhere)

Helper methods

ackage/folder)

`pr` s and its subclasses)

`private` (only this class)

```java
public class Car
{
    public void repair()
    {
        runDiagnostics();
        disassembleEngine();
        repairBrokenParts();
        reassembleEngine();
        runDiagnostics();
    }
    private void runDiagnostics() { /*...*/ }
    private void disassembleEngine() { /*...*/ }
    private void repairBrokenParts() { /*...*/ }
    private void reassembleEngine() { /*...*/ }
}
```

```
public class SomeCarApplicationProgram
{
    public static void main(String[] args)
    {
        Car  c = new Car();
        c.repair
        c.disassembleEngine();
        c.repairBrokenParts();
    }
}
```

```
public class SomeCarApplicationProgram
{
    public static void main(String[] args)
    {
        Car  c = new Car();
        c.repair();
        c.disassembleEngine();
        c.repairBrokenParts();
    }
}
```

public, so ok to call

```
public class SomeCarApplicationProgram
{
    public static void main(String[] args)
    {
        Car  c = new Car();
        c.repair();
        c.disassembleEngine();
        c.repairBrokenParts();
    }
}
```

private – can’t call directly like this

Getters and Setters:

methods that provide controlled access to private internal parts of an object

Object attributes are easier to understand and use
Attributes are protected from external/unknown changes
We are following proper and robust coding style

Get Methods

public access
return type matching attribute’s type
name matching attribute’s name
code returning attribute’s value

Get Methods in Java

```
public class Patient
{
    private String   name;
    private int      age;
    private float    height;
    private char     gender;
    private boolean  retired;

    // Get methods for name, age, height,
    // gender and retired attributes
    public String getName() { return name; }
    public int getAge() { return age; }
    public float getHeight() { return height; }
    public char getGender() { return gender; }
    public boolean isRetired() { return retired; }
}
```

Set Methods

public access
void return type
name matching attribute’s name
a parameter matching attribute’s type
code giving the attribute a value

Set Methods in Java

```
// Set method for name attribute
public void setName(String n)
{
    name = n;
}

// Set method for age attribute
public void setAge(int a)
{
    age = a;
}
```

```
// Set method for height attribute
public void setHeight(float h)
{
    height = h;
}

// Set method for gender attribute
public void setGender(char g)
{
   gender = g;
}

// Set method for retired attribute
public void setRetired(boolean r)
{
   retired = r;
}
```

So why did we write all this extra code again?

Abstraction
Control over what can be accessed and how
Easier maintenance of bigger systems (decoupling)
Code reuse

Readability and Expressiveness

Always think about how your classes will be used in the future, and use good method names.

Write your classes that make the rest of your program easier to write.

Always think about how your classes will be used in the future, and use good method names.

Write your classes that make the rest of your program easier to write.