I found this funny example in terms of loose coupling:

Source: *Understanding dependency injection*

Any application is composed of many objects that collaborate with each other to perform some useful stuff. Traditionally each object is responsible for obtaining its own references to the dependent objects (dependencies) it collaborate with. This leads to highly coupled classes and hard-to-test code.

For example, consider a Car object.

A Car depends on wheels, engine, fuel, battery, etc. to run. Traditionally we define the brand of such dependent objects along with the definition of the Car object.

Without Dependency Injection (DI):

```
class Car{
  private Wheel wh = new NepaliRubberWheel();
  private Battery bt = new ExcideBattery();

  //The rest
}
```

Here, the Car object *is responsible for creating the dependent objects.*

What if we want to change the type of its dependent object - say Wheel - after the initial NepaliRubberWheel() punctures? We need to recreate the Car object with its new dependency say ChineseRubberWheel(), but only the Car manufacturer can do that.

Then what does the Dependency Injection do for us...?

When using dependency injection, objects are given their dependencies *at run time rather than compile time (car manufacturing time).* So that we can now change the Wheel whenever we want. Here, the dependency (wheel) can be injected into Car at run time.

After using dependency injection:

Here, we are **injecting** the **dependencies** (Wheel and Battery) at runtime. Hence the term : *Dependency Injection.* We normally rely on DI frameworks such as Spring, Guice, Weld to create the dependencies and inject where needed.

```
class Car{
  private Wheel wh; // Inject an Instance of Wheel (dependency of car) at runtime
  private Battery bt; // Inject an Instance of Battery (dependency of car) at runtime
  Car(Wheel wh,Battery bt) {
     this.wh = wh;
     this.bt = bt;
  }
  //Or we can have setters
  void setWheel(Wheel wh) {
     this.wh = wh;
  }
```

}

The advantages are:

- decoupling the creation of object (in other word, separate usage from the creation of object)
- ability to replace dependencies (eg: Wheel, Battery) without changing the class that uses it(Car)
- promotes "Code to interface not to implementation" principle
- ability to create and use mock dependency during test (if we want to use a Mock of Wheel during test instead of a real instance.. we can create Mock Wheel object and let DI framework inject to Car)