Context

1. Building a system (like a car, a website, a database) is hard, so we break it down into small pieces.
2. We don't want to redo previous work (**DRY principle**, or the "don't repeat yourself" principle)

Object-oriented programming (OOP) is a technique to break down problems / systems into smaller pieces and save us time in the process. It is also a way to customize already-implemented pieces of code (sometimes).

Definitions

- **Class** - a definition or a type. Has variables and methods (it knows things and can do things).
- **Object** - a specific instance of a class

Example: I describe a class called "student", and every student has a "grade" and a "grade level" and a "name". The specific student "Nina" is an object.

A **class** is a description, an **object** is a specific version of the description.

When you create an **object**, you are **initializing** or **instantiating** a class. That **object** is of **type** class.

Definition

- **Subclass** - a class that **inherits** from another class. It adds things to an already-created class. It is of **type** itself and it's superclass.
- **Superclass** - the class that a subclass inherits from

A subclass has all the variables and methods from the superclass

Classes in Python

In [11]:
```
class Student(object):
    def __init__(self, name_to_set="", year_to_set=0, grade=100):
        self.name = name_to_set
        self.year = year_to_set
        self.grade = grade

    def letter_grade(self):
        if self.grade >= 90:
            return "A"
        elif self.grade >= 80:
            return "B"
        elif self.grade >= 70:
            return "C"
        elif self.grade >= 65:
            return "D"
        else:
            return "F"
```

- In Python, every class is a subclass of the master class **object**
- By convention, Python class names should be `CamelCase` and method names should be `names_with_underscores_for_spaces`
- I have "default arguments" in my function parameters

Subclasses in Python

In [34]:
```
class UniversityStudent(Student):
    def __init__(self, name_to_set="", year_to_set="", uni=""):
        super(UniversityStudent, self).__init__(name_to_set, year_to_set)
        self.university = uni

    def signature(self):
        return self.name + ", " + self.university + " " + str(self.year)
```

In [40]:
```
n = UniversityStudent("Kyle Hannon", 2013, "MIT")
print n.name
print n.grade
print n.year
print n.signature()
print isinstance(m, UniversityStudent)
print isinstance(m, Student)
print isinstance(n, UniversityStudent)
print isinstance(n, Student)
```

```
Kyle Hannon
100
2013
Kyle Hannon, MIT 2013
False
True
True
True
```

Writing for OOP

- Let's assume you already decided on what class you will write
- First, design. What variables will you need? What methods will you need?
- What behavior do you want it to have? (how do you want to test it?)