Introduction to Object Oriented Programming

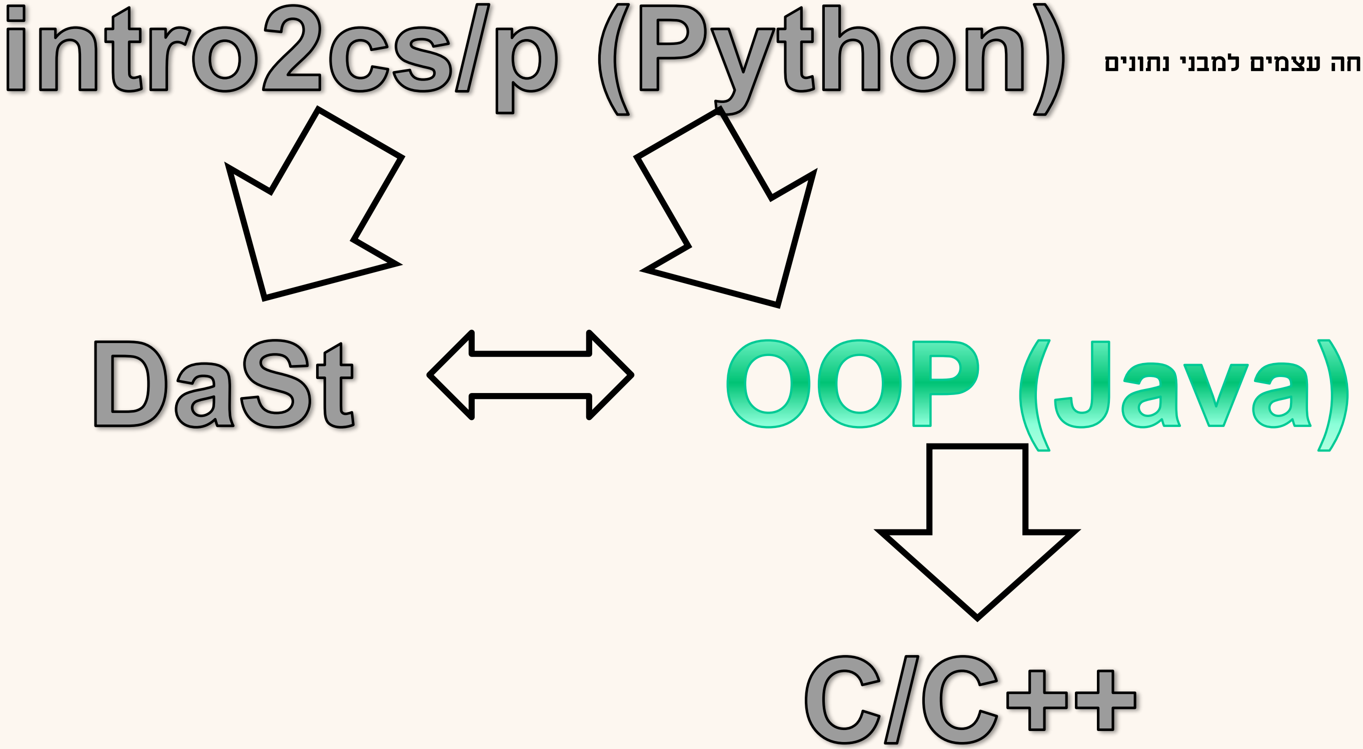
Roy Schwartz, The Hebrew University (67125)

Lecture 1:

Course Introduction

Lecture 1a: Overview

- Introduction to the course
- What is object oriented programming?
- Objects
- Classes
- Types
- Constants



intro2cs/p (Python)

DaSt

OOP (Java)

C/C++

הממשק בין מונחה עצמים למבני נתונים

Course Goals

תכנות נכון המבוסס על מבני
נתונים מתאימים

- **OOP**:
  - Object Oriented Programming principles and concepts
  - Learn basic Object Oriented design including basic design patterns

- **DaSt**:
  - Implement data structures based on ideas from the Data-Structures course
  - Learn to use and estimate complexity of existing data structures in java collections

Course Goals

ג׳אווה כשפה אינדיווידואלית
ג׳אווה כשפת מונחה עצמים

- **Java Programming**:
  - Familiarize with the java programming language
  - Get acquainted with important and useful **java specific** principles, as well as general **programming** principles

Course Format

- Weekly online lecture
  - Organized in ~4-8 parts
  - Each part ~5-15 minutes long

Course Syllabus

- **Introduction to java** (Lectures 1-2)
- **Polymorphism and Basic Design** (Lectures 3-5)
- **Core Topics in java** (Lectures 6-7)
- **Modularity and Advanced Design** (Lectures 8-9)
- **Advanced Topics** (Lectures 10-13)

Lecture 1b: Overview

- Introduction to the course
- What is object oriented programming?
- Objects
- Classes
- Types
- Constants

Properties of Good Program

- Working!
  - Meets the requirements
- Easy to learn and use
- Fast & efficient
- Fail-safe
- Fool-safe
- Hard-to-hack
- Compatible

- Fast to code
- Easy to test & debug
- Easy to understand

  (by other **team members** or by **same programmer** in the future)
- **Can be reused**
- **Easy to update/upgrade**

Why Object-Oriented?

- Building large systems
  - Many components that **share pieces of code**
  - Many **interdependencies**
  - Frequent **changes in requirements**

- ➢ Easy to understand
- ➢ **Can be reused**
- ➢ **Easy to update/upgrade**

Basic OOP Concepts

1. Objects and Classes
2. Encapsulation
3. Inheritance
4. Polymorphism
5. Genericity

Basic OOD(esign)

1. Modularity
2. Design Patterns

Object-Oriented Programming

- A programming paradigm, in which a program can be viewed as a set of interactions between **objects**
- An alternative to *procedural programming*
  - In which a program is a list of **procedures**
- Used in many programming languages
  - C++, PASCAL, Python
- The main programming principle in many languages
  - ***java***, C#

ההבדל בין תכנות מונחה עצמים
לבין תכנות פרוצדורלי

Lecture 1c: Overview

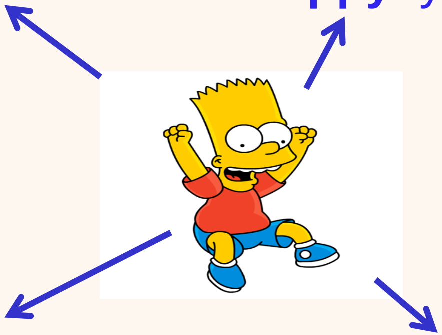
- Introduction to the course
- What is object oriented programming?
- Objects
- Classes
- Types
- Constants

What is an Object?

- Real-world objects share two characteristics: They all have *state* and *behavior*

Name: Bart

isHappy: yes



Hair color: Yellow

jump

Name: Pluto

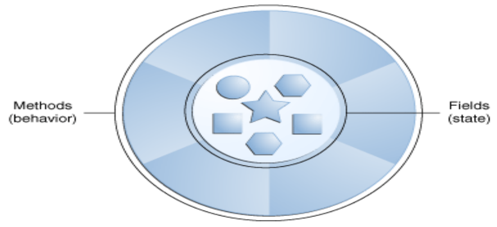
isHappy: yes

Number of siblings: 4

wag tail

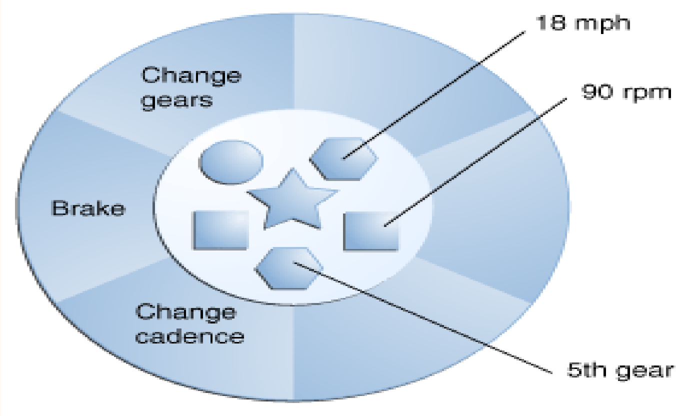
Software Objects

- Software objects can
  - Hold **information** (internal *states* – “*data members*”)
  - Perform actions (external *behavior* –“*methods*”)



Object Example

- A bicycle object



Object-oriented vs. Procedural Programming

- Procedural Programming
  - get_name(child)
  - wag_tail(dog)
  - get_length(string)
  - equals(dog1, dog2)

- Object-oriented Programming
  - child.getName()
  - dog.wagTail()
  - string.getLength()
  - dog1.equals(dog2)

על מי האחריות לבצע פעולה?
על אובייקט או על פונקציה?

OOP: Motivating Example

Animals

כל המשתנים פתוחים לכל חלקי הקוד

- Procedural Programming
  - bark(dog)
  - meow(cat)
  - moo(cow)
  - …

- Object-oriented Programming
  - dog.makeSound()
  - cat.makeSound()
  - cow.makeSound()

Same Code

- ➢ Easy to understand
- ➢ Easy to update/upgrade

Lecture 1d: Overview

- Introduction to the course
- What is object oriented programming?
- Objects
- Classes
- Types
- Constants

Some Objects are Similar

- In the real world, many individual objects are of the same type
  - Many different dogs exist
  - Each dog has 4 legs, can wag its tail, etc.
- Different objects of the same type may have different states
  - Pluto, Guffy, and Rex are all different dogs
  - They have different **names**, different **parents** and **siblings**, etc.
- But all can perform the same actions
  - Run, bark, wag tail…

Classes

- Software classes are used to define groups of objects
- These groups share the same *types of members* (i.e., possible *states*) and the same *methods* (i.e., *behavior*)
- Objects of a given class (denoted *instances*) provide concrete values to each of the data members
  - Goofy (a dog *object*) is an *instance* of the class Dog
  - myBike is an *instance* of the class Bicycle

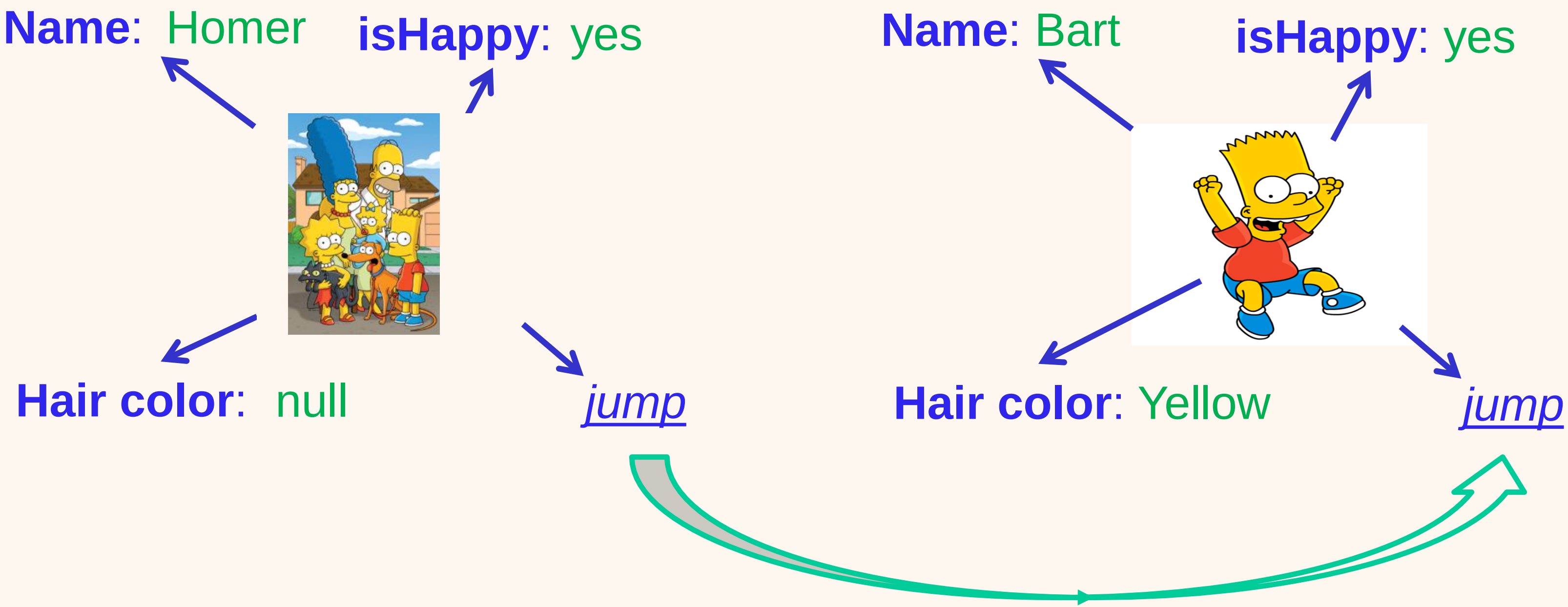
Data Members

- Data members are variables defined by a class
- All objects of a given class have the exact same set of data members
  - A dog’s name, the brand of a bicycle…
- Different objects give (potentially) different values to the same data member
  - One dog is called Goofy, the brand of one bike is BMX…

Methods

- Methods are functions associated with a specific class
- Every object of a given class has the same set of methods
  - All Dog objects can run, bark, etc.
  - All bikes can break, change gear, etc.
- Methods can access the data members of a given object
  - Dog.getAllSiblings() behaves differently for dogs with a different number of siblings
- The procedural part of java lies in methods

Class Example



Name: Homer

isHappy: yes

Hair color: null

jump

Name: Bart

isHappy: yes

Hair color: Yellow

jump

Class Code Example

Bicycle.java

```
class Bicycle {
        /* Data members */
        int speed = 0;
        int gear = 1;
        String brand;

        // Methods
        void changeGear(int newValue) {
                gear = newValue;
                return;
        }
```

```
        // Other methods
        int speedUp(int increment) {
                speed = speed + increment;
                return speed;
        }

        void break() {
                speed = 0;
                return;
        }
        …
}
```

Creating New Objects

- Classes define how each of their objects (instances) look like
  - What are their members and methods

- Each class defines a special method (or methods) called *constructor(s)* that allow(s) the creation of new objects

Constructors

- Constructors are methods used for assigning values to the data members of the new object
  - The bicycle brand, initial speed, etc.
- Java constructors have several properties:
  - They use the same name as the class
  - They have no return value
  - They can get a set of parameters, just like any other method (including no parameters)

Constructor Example

Bicycle.java

```
class Bicycle {
        /* Data members */
        int speed = 0;
        int gear;
        String brand;

        /* Constructor */
        Bicycle(String myBrand, int newGear) {
                brand = myBrand;
                gear = newGear;
        }
```

```
        /* Methods */
        void changeGear(int newValue) {
                gear = newValue;
        }

        int speedUp(int increment) {
                speed = speed + increment;
                return speed;
        }

        …
}
```

Using Constructors

BicycleDemo.java

```
class BicycleDemo {
        public static void main(String[] args) {
                // Create two different Bicycle objects
                Bicycle bike1 = new Bicycle(“BMX”, 1);
                Bicycle bike2 = new Bicycle(“newbike”, 2);

                // Invoke methods on those objects
                bike1.speedUp(10);
                bike1.changeGear(2);
                System.out.println(bike1.gear+“, ”+bike1.speed);
                bike1.changeGear(3);
                System.out.println(bike1.gear+“, ”+bike1.speed);
                bike2.speedUp(10);
                bike2.break();
                System.out.println(bike2.gear+“, ”+bike2.speed);
        }
}
```

bike1 and bike2 belong to the Bicycle class

Output:

```
2, 10
3, 10
2, 0
```

Classes vs. Objects

Memory Issues

- Only one copy of the **class** exist
  - Memory to store methods is **allocated once**
- For each class, many **objects** potentially exist
  - Memory is allocated **for each of them**
- Each object belongs to exactly one class*

* This is not accurate. See later in course.

Lecture 1e: Overview

- Introduction to the course
- What is object oriented programming?
- Objects
- Classes
- Types
- Constants

Types

- A java variable can either be
  - a **reference** (to an object)
  - or a **primitive** (**int**, **double**, **char**…)

Primitives

- Java defines several types of primitives that hold the most basic data types
  - **int** – an integer (5, 7, -1, 0, …)
  - **double** – a floating point number (5.0, -2.6, 0.0, …)
  - **char** – a single character (‘a’, ‘b’, ‘#’, ‘?’, …)
  - **boolean** – a boolean variable (**true**, **false**)
- Each primitive of a given type requires the same amount of memory

Reference

- A reference is not an actual object, but something that **points** to an object
- Each reference has a type, which is the object’s class name*
  - **Dog** myDog, **Bicycle** myBike, …

* See alternatives later in the course

Reference vs. Content

- The following line contains two parts, separated by the ‘=’ sign:

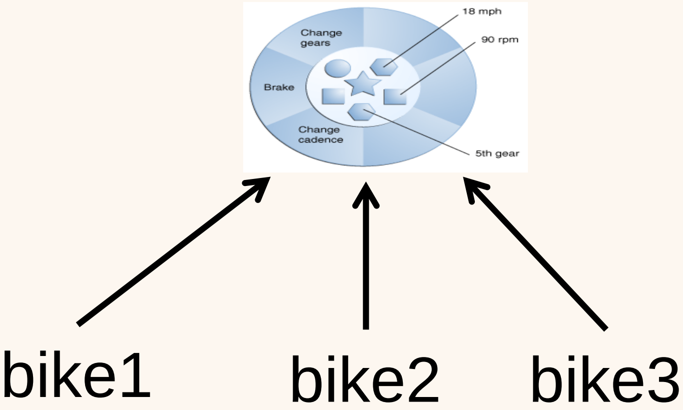
Bicycle bike1 = ***new*** *Bicycle(1);*

- The first part (*Bicycle bike1)* defines a new **reference** to an object of type *Bicycle*

- The second part (***new*** *Bicycle(1)*) defines **content**
  - A concrete object

Reference Example

- The creation of new references doesn’t waste much memory
  - *Bicycle bike1 = **new** Bicycle(1);*
  - *Bicycle bike2 = bike1;*
  - *Bicycle bike3 = bike1;*
  - …



bike1 bike2 bike3

Content

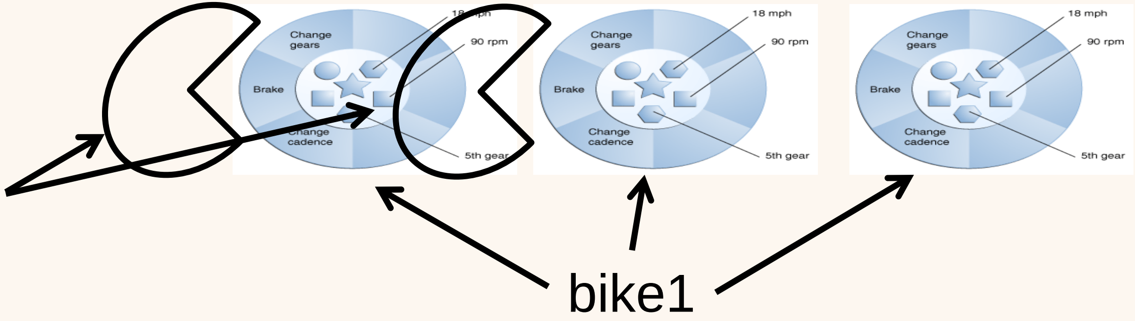
- Calling a constructor (using the **new** keyword) creates a new object
  - Each call requires more memory

Content Example

Bicycle bike1 = ***new*** *Bicycle(1);*

bike1 = ***new*** *Bicycle(1);*

bike1 = ***new*** *Bicycle(1);*



Garbage Collector

bike1

- The reference-content distinction has other implications
    - See later in this course

The String Class

- The most common java class
- Although it is not a primitive, can be initialized using the ‘=’ sign
  - String myString = “hello”;
- Has many important useful methods
  - length(), charAt(…),
- Is **immutable**
  - Can’t change content of string (e.g., change 1st char to ‘y’)
  - More on this to come

Lecture 1f: Overview

- Introduction to the course
- What is object oriented programming?
- Objects
- Classes
- Types
- Constants

Introduction to Object-oriented Design

- TMTOWTDI
  - There's more than one way to do it
- We are trying to build pieces of software that yield **good programs**
  - Working, extensible, easy to debug, efficient, etc.
- There is hardly ever a perfect solution
  - A good design is one in which the pros out-weight the cons
  - Obtaining one usually requires expertise

Design Case Study:
Constants

- Many programming languages (including *java*) allow the creation of constant variables
  - These are variables that don’t allow changing their values
  - Their value is set once at the creation of the object
- In java, you add the keyword **final** before the variable type
  - **final int** myInt = 5;
  - **final** String str = “hello”;
  - **final** Bicycle myBike = **new** Bicycle(1);

Why Use Constants?

- Some properties of an object should never be changed
  - A dog’s name
  - A bike’s maximal gear
- We should decide, at design time, which properties (i.e., data members) should remain the same throughout the lifetime of the object

Constants Example

Dog.java

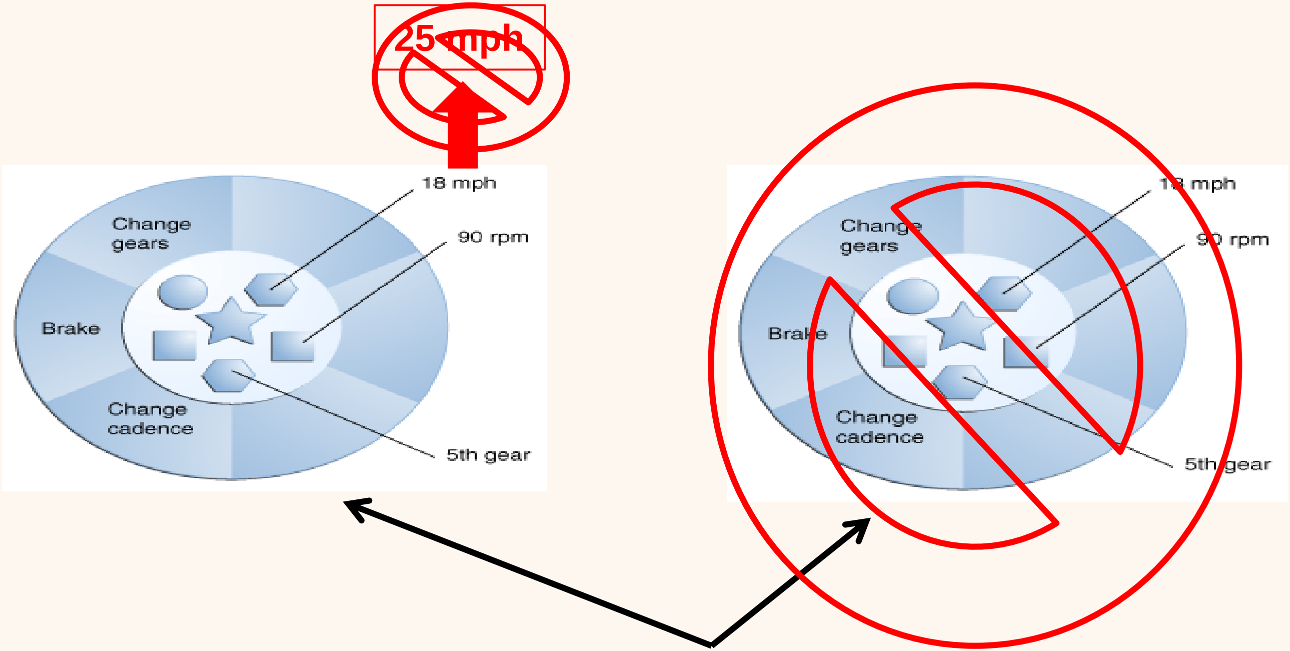
```
class Dog {
    /* Data Members*/
    final String name;
    int nSiblings;

    /* Constructors */
    Dog (String dogName, int nDogSiblings) {
        name = dogName;
        nSiblings = nDogSiblings;
    }

    …
```

```
    public static void main(String args[]) {
        Dog myDog = new Dog(“pluto”, 5);
        myDog.nSiblings = 3;      // ok
        myDog.name = “goofy”;     // Error!
    }
}
```

Constant vs. Immutable

- Reminder: the String class is immutable
  - String s = “hello”;
  - Impossible to change the content of that String **object** (e.g., you **cannot** run a code like s.charAt(0) = ‘y’)
- It is possible to assign a different object to the *s* reference
  - s = “goodbye”;
- This is impossible when *s* is declared **final**
  - **final** String s = “hello”;
  - s = “goodbye”;   **// Error**

Constant vs. Immutable



final Bicycle bike1

Why Force It?

- If someone wants to change a dog’s name, why should we prevent her from doing it?
- A major issue in design is prevention instead of cure
  - If we design a dog class such that its name should never change, we should **prevent users from changing it (fool-safe)**
  - Users can be either us, or other programmers that use our code
- When someone uses our code in the wrong way, **bugs** occur
  - This may not be our fault, but this is **our problem**

So far…

- Writing a Good Program
  - Works, fast, extensible, …

- Object-oriented Programming
  - Class vs. object
  - Constructors
  - Reference vs. content
  - Constants

Next Week

- Scope
- Instance vs. static
- Minimal API
- Information Hiding