Introduction to Programming II

Processing Advanced

Creative Technologies I Summer term 2019

FILMUNIVERSITÄT
BABELSBERG
KONRAD WOLF

Angela Brennecke I Prof. Dr.-Ing.
Audio & Interactive Media Technologies
Film University Babelsberg KONRAD WOLF

Contents

- Object-oriented programming
- Classes and objects

Classes & Objects

- To create a custom data structure or object, you have to define its properties & methods in a corresponding class

- Classes are the blueprint for the actual objects

- Based on the class definition various different object instances can be created or instantiated

Classes & Objects

- In order to actually define & implement an object we have to understand the notion of a “**class**”

- A **class** is the blueprint of an object

- Classes specify
  - the properties of an object (i.e., class variables)
  - the methods of an object (i.e., class functions)

Classes & Objects

- The class specifies the type of the object
  - **class name**
  - **constructors** (to instantiate the object)
  - **properties** (e.g. primitive or object variables)
  - **methods** (that represent object functions)

Class Name & Constructor

- The class name will always be used when an object of the class is instantiated in the code

- The class name and the constructor must be the same

- A class can have numerous different constructors

```
3  // Class name "Door"
4  class Door
5  {
6    // ...
7
8    // Class constructors -- these functions get called
9    //                       when the object is instantiated
10
11   Door() {}
12
13   Door(float w, float h)
14   {
15     this.w = w;
16     this.h = h;
17   }
18
19     Door(float x, float y, float w, float h)
20   {
21     this.x = x;
22     this.y = y;
23     this.w = w;
24     this.h = h;
25   }
26
27   // ...
28 }
```

Class Name & Constructor

- In the main routine a Door object gets instantiated by calling one of its' constructors with new & assigning the object to the object variable "door1"

```
5  void setup()
6  {
7    size(200, 350);
8    background(200);
9    smooth();
10
11   // Create new Door object by
12   // calling its constructor
13   Door door1 = new Door(100, 250);
14
15   // ...
16 }
```

```
3  // Class name "Door"
4  class Door
5  {
6    // ...
7
8    // Class constructors -- these functions get called
9    //                       when the object is instantiated
10
11   Door() {}
12
13   Door(float w, float h)
14   {
15     this.w = w;
16     this.h = h;
17   }
18
19     Door(float x, float y, float w, float h)
20   {
21     this.x = x;
22     this.y = y;
23     this.w = w;
24     this.h = h;
25   }
26
27   // ...
28 }
```

Class Properties

- Class properties are usually defined at the top of the class after the class name & before the constructors
- They can be primitive data types or even other objects

```
// Class name "Door"
class Door
{
  // Class properties -- the variables of the class

  float x = 15;
  float y = 15;
  float w = 30;
  float h = 30;

  boolean hasKnob = false; // No knob by default!

  final static int KNOBRIGHT = 0; // "final static" defines constant variables, they do not change
  final static int KNOBLEFT = 1;

  int knobLocation = 0; // default knoblocation (if we have a knob) is KNOBLEFT


  // Class constructors -- these functions get called
  //                       when the object is instantiated
  // ...
}
```

Class Properties

- The actual value of a property is not fixed,
  it may differ from object to object
- In the main routine, the properties can be directly manipulated

```
// Class name "Door"
class Door
{
  // Class properties -- the variables of the class

  float x = 15;
  float y = 15;
  float w = 30;
  float h = 30;

  boolean hasKnob = false; // No knob by default!

  final static int KNOBRIGHT = 0; // "final static" defin
  final static int KNOBLEFT = 1;

  int knobLocation = 0; // default knoblocation (if we ha


  // Class constructors -- these functions get called
  //                       when the object is instantiate
  // ...
}
```

```
void setup()
{
  size(200, 350);
  background(200);
  smooth();

  // Create new Door object by
  // calling its constructor
  Door door1 = new Door(100, 250);
  door1.hasKnob = true;

  Door door2 = new Door(150, 15, 10, 100);
  door2.hasKnob = false;
  // ...
}
```

Object-Oriented Programming

Class Methods

- Class methods are functions that add functionality to the class and objects

- When a function of an actual object is invoked, it will follow the same instructions for every individual object

- However, the results will depend on the individual property values set per object!

```
// Class name "Door"
class Door
{
  // ...

  // Class methods

  // draw the door
  void drawDoor()
  {
    rect(this.x, this.y, this.w, this.h);

    if (this.hasKnob == true)
    {
      drawDoorKnob();
    }

  }

  void drawDoorKnob()
  {
    float knobsize = this.w/10;  // the knob width is 1/10 of the door width

    if (knobLocation == 0)
    {
      //right side
      ellipse(x+w-knobsize, y+h/2, knobsize, knobsize);
    } else
    {
      //left side
      ellipse(x+knobsize, y+h/2, knobsize, knobsize);
    }
  }

  // ...
}
```

Class Methods

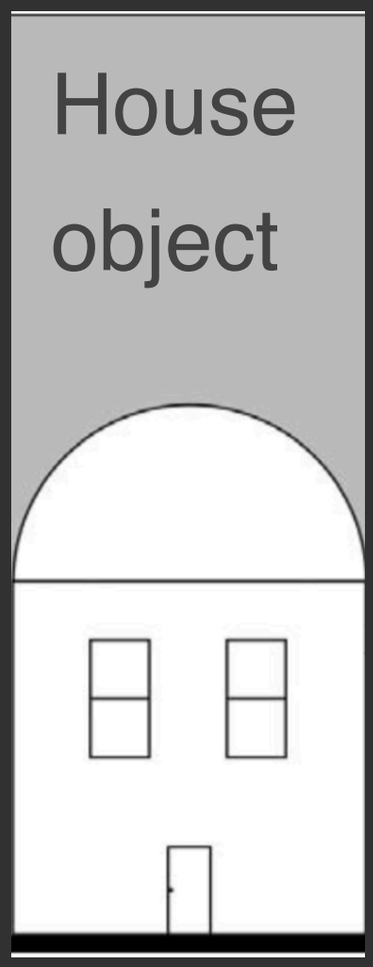
- In the main routine, class methods are called per object
- The results depend on the settings of the properties

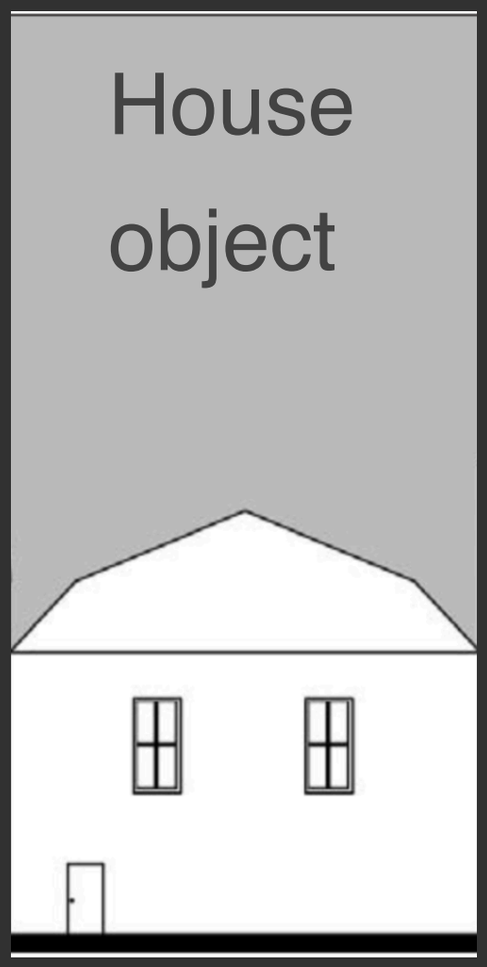
```
// Class name "Door"
class Door
{
  // ...

  // Class methods

  // draw the door
  void drawDoor()
  {
    rect(this.x, this.y, this.w, this.h);
```

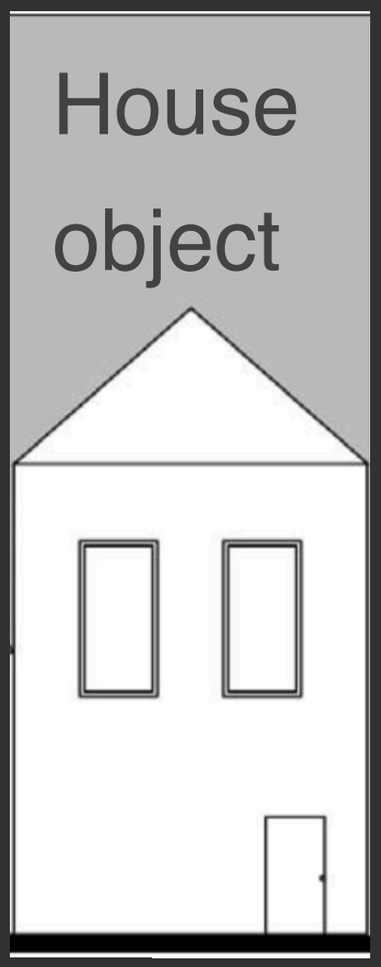
```
void setup()
{
  size(200, 350);
  background(200);
  smooth();

  // Create new Door object by
  // calling its constructor
  Door door1 = new Door(100, 250);
  door1.hasKnob = true;
  door1.drawDoor();

  Door door2 = new Door(150, 15, 10, 100);
  door2.hasKnob = false;
  door2.setKnobLocation(Door.KNOBLEFT);
  door2.drawDoor();
  //
}
```

Classes & Objects

```
class House
{
    // …
}
```







one class — many (individual) object instances

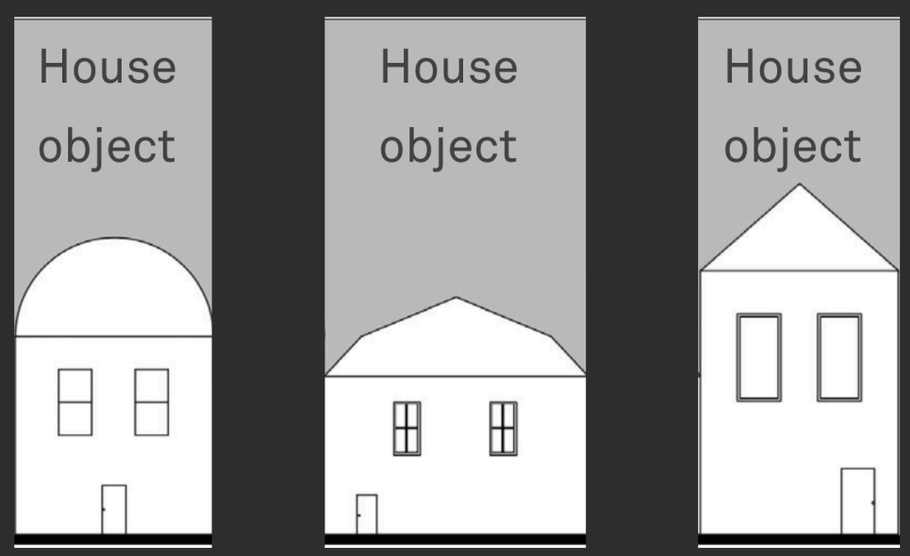
Images credit: [IGr07], p. 396.

Object Instantiation

- Object instantiation refers to the actual creation of an object that is of type class

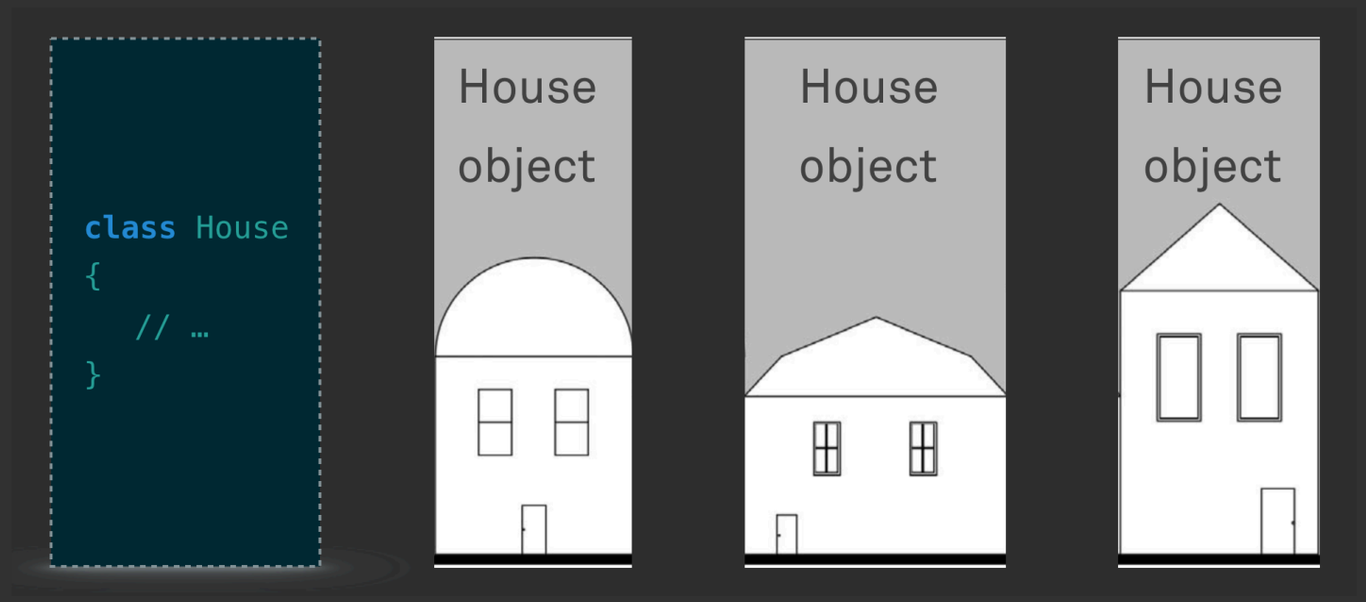
- Instead of “**objects**”, you can also speak of the “**instances**” of a class that are created

```
class House
{
   // …
}
```



Object Instantiation

- The **class** is the abstract specification of an object type
- The **object** or **instance** is the actual component or data type that is being processed in the application



Primitive & Object Variables

- **Primitive variables** represent one value that corresponds to the **data type** that defines the primitive

- **Object variables** represent the object correspond to the type / class of the
  - they can be used to request to object properties
  - they can be used to call object methods

Primitive & Object Variables

lecture02_objectVariables.pde

```
1  void setup ()
2  {
3    size(400, 400);
4    background(255);
5    noStroke();
6
7    // instantiating primitive variables of type integer, float, ..
8    int exampleInteger = 45;
9    float exampleFloat = 65.0f;
10   boolean isReady = false;
11
12
13   // instantiating a PImage object
14   PImage myImage = new PImage();
15   // calling the PImage Class method "isLoaded" of the myImage object
16   isReady = myImage.isLoaded();
17
18
19   // instantiating a custom object of type myRect
20   myRect aCustomRectangle = new myRect();
21   // accessing and changing the object's property "positionX"
22   aCustomRectangle.positionX = width/2;
23   // calling the myRect Class method "draw" of the aCustomRectangle object
24   aCustomRectangle.draw();
25 }
```

Wrap up

- Object oriented programming is a new programming paradigm that requests from the developer, you, to break down a task or problem into its individual components — the **objects** of the whole systems / application
- The specification of an object happens in a **class**
- Class require to define class constructor(s), properties and finally methods that allow to describe functionality

Practical Exercise

Practical Exercise

- We now have two options:
  - You can already start to decide and work on an individual creative coding project, for example, create your own custom 2 scene, add a character and let the character move across the scene
- We review the retro „snake game“ and implement it in processing
  - You have to come up with a system design on paper
  - We will review the design and start with the implementation
  - You can customize the game and turn it into your cc project

Creative Coding Project

Angela Brennecke I Prof. Dr.-Ing.
Audio & Interactive Media Technologies
Film University Babelsberg KONRAD WOLF

Creative Coding Project

- Homework assignment
- Goals and scope to be discussed & defined in class

Next Session: Tuesday 13 May, 16:00

Bibliography

Bibliography

- [IGr07] Ira Greenberg (2007): **Processing: Creative Coding & Computational Art**. Friends of ED/Apress Press. Berkley, CA.
- [JNo09] Joshua Noble (2009): **Programming Interactivity. A Designer's Guide to Processing, Arduino & openFrameworks.** O'Reilly Media Inc. Sebastopol, CA.
- [RMa09] Robert C. Martin (2009): **Clean Code**. Upper Saddle River, NJ: Prentice Hall.
- https://github.com/shiffman | last access 2019/02
- https://www.funprogramming.org | last access 2019/02