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

- Introduction to the course
- Recap & warm-up exercises
- Object-oriented programming
- Project work

Introduction to the Course

- Dates & times of the lecture
  - Tuesday 16 April 10:00 - 17:00
  - Tuesday 30 April 10:00 - 13:00
- 1 + 1 ECTS
  - 1 per attending the lecture & exercises
  - 1 per participating in a creative coding project

Learning Objectives

- Improve coding skills towards software design
- Understand object-oriented programming paradigm
- Apply game idea to OOP & implement it
- Develop an own creative coding project

About the Lecturer

Angela Brennecke

2004	Dipl.-Ing. of Computational Visualistics
2009	Dr.-Ing. of Computer Graphics
2009-2014	Native Instruments GmbH | SWDev, PJM, TeamLead
2015	Creative work with songwriting & composition
2016-2017	RBB Innovation Projects | Project engineer
since **2017**	Professor of Audio & Interactive Media Technologies

About the Audience

- Who are you?
- What do you want to take away from this course?

Recap & Warm-up

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

Recap & Warm-up

- Programming modes
- Variables
- Functions
- Naming conventions

Programming Modes

- Processing has two programming modes

```
// basic mode

size(600, 600);
background(255);
smooth();

stroke(145);
strokeWeight(5);
fill(0, 200, 0, 100);
rect(200, 200, 150, 200);
```

```
// continuous mode

void setup( ) {
  size(600, 600);
  background(255);
  smooth();
}

void draw( ) {
  stroke(145);
  strokeWeight(5);
  fill(0, 200, 0, 100);
  rect(200, 200, 150, 200);
}
```

Programming Modes

- Processing has two programming modes

- Basic mode uses
  - instructions & commands are executed **linearly**
  - no complex functionality like functions or classes

- Continuous mode uses
  - instructions & commands are executed **non-linearly**
  - more complex functionality like functions & classes

Programming Modes

- Continuous mode provides two basic functions that allow to add custom functions & classes

- void setup( )
  - called only once at program startup
  - used to initialize variables
- void draw( )
  - called continuously throughout program runtime
  - used to do animations, screen refreshing, drawings, …

Variables

- Processing is a statically typed language
- Strict typing is mandatory

```
int aSpeed = 6;
boolean isReached = aSpeed  // !! Compiler Error !!
```

- Primary advantage of statically typed languages is that code can be executed faster

Variables

- Processing supports two types of variables
  - **primitive** variables
  - **object** variables (we'll look at them later)

- Main difference is what kind of **data type** can be associated with them & how they are stored in the computer's memory

Variables

What else do you know about using variables in Processing?

```
1
2  int BGColor;
3  int bgColor = 155;
4
5  void setup( ) {
6    size(500, 500);
7    background(bgColor);
8    BGColor = 90;
9  }
10
11 void draw( ) {
12   stroke(BGColor);
13   line( 0, 0, mouseX, mouseY);
14 }
```

Variables

- declared
- initialized
- case sensitive
- mutable
- global
- local
- system defined

```
1
2  int BGColor;
3  int bgColor = 155;
4
5  void setup( ) {
6    size(500, 500);
7    background(bgColor);
8    BGColor = 90;
9  }
10
11 void draw( ) {
12   stroke(BGColor);
13   line( 0, 0, mouseX, mouseY);
14 }
```

Functions

- Functions organize code into reusable blocks
- They add structure and flexibility to a program
- When writing custom functions, void setup( ) and thus continuous mode is mandatory

```
1  // continuous mode
2
3  void setup( ) {
4    size(600, 600);
5    background(255);
6    smooth();
7  }
8
9  void draw( ) {
10   stroke(145);
11   strokeWeight(5);
12   fill(0, 200, 0, 100);
13   rect(200, 200, 150, 200);
14 }
```

Functions

```
// based on Sketch from Ira Greenberg's book, p. 97

void setup( )
{
  size(600, 600);
  background(255);

  boolean isDrawn = false;

  for (int i=0; i<100; i++) {
    isDrawn = drawRectangle( random(width), random(height), random(200), random(200) );
    println( isDrawn );
  }
}

boolean drawRectangle(float x, float y, float w, float h )
{
  rect(x, y, w, h);
  return true;
}
```

Functions

```processing
// based on Sketch from Ira Greenberg's book, p. 97

void setup( )
{
  size(600, 600);
  background(255);

  boolean isDrawn = false;

  for (int i=0; i<100; i++) {
    isDrawn = drawRectangle( random(width), random(height), random(200), random(200) );
    println( isDrawn );
  }
}

boolean drawRectangle(float x, float y, float w, float h )
{
  rect(x, y, w, h);
  return true;
}
```

- Custom functions in Processing can be declared using
  - parameters
  - return values
- If they don't return a value, void must be declared

Functions

```
1  // based on Sketch from Ira Greenberg's book, p. 97
2
3  void setup( )
4  {
5    size(600, 600);
6    background(255);
7
8    boolean isDrawn = false;
9
10   for (int i=0; i<100; i++) {
11     isDrawn = drawRectangle( random(width), random(height), random(200), random(200) );
12     println( isDrawn );
13   }
14 }
15
16 boolean drawRectangle(float x, float y, float w, float h )
17 {
18   rect(x, y, w, h);
19   return true;
20 }
```

- Function names should clearly explain what the function does
- Functions should do one thing
- Parameter lists should be short

Naming Conventions

- Some words on naming conventions
- Think of code as prose — something you write not only for your own pleasure but for somebody else to read, understand & potentially work with

Naming Conventions

There are no fixed rules, but …

- Function & variable names
  - should support the overall understanding of the code

- should be easy to read
- should be consistent
- should clearly indicate what they intent

Naming Conventions

```
1 int etd = 35;              // elapsed time in days
2 int fy = 15;               // number of fixed years
3 float temp = 15.5f;        // temperature
4 boolean b1 = true;         // indicates whether time has passed
5 boolean b2 = false;        // indicates whether temperature level was reached
```

Which set of variables tells you more about what they are used for?

```
 9 int iElapsedTimeInDays = 35;
10 int iFixedYears = 15;
11 float fTemperature = 15.5f;
12 boolean isPassed = true;
13 boolean isReached = false;
```

Naming Conventions

```
35 int checkNum()
36 {
37   if (n < maxi) {
38     return -1;
39   }
40   else {
41     return 1;
42   }
43 }
```

```
35 boolean hasTakenDamage()
36 {
37   return playerHealth < maxHealth;
38 }
```

Which function tells you more about what it is used for?

Naming Conventions

- Following „Uncle Bob“, you can always ask yourself these questions when naming a function or a variable:
  - Does the name reveal the intentions?
  - Is the name meaningfully distinguished from others?
  - Is the name consistent with the rest of the code?

```
int getNumOfYears( ) { … }
int getHeight( ) { … }
…
// vs
int getNumOfYears( ) { … }
int fetchHeight( ) { … }
…
```

```
int yearsNum = 10;
int numHeight = 2;
…
// vs
int numYears = 10;
int numHeight = 2;
…
```

Naming Conventions I Take away

- Coding is communicating — comprehensibility comes first

- Function & variable names should be comprehensible and reveal their meaning and purpose to the reader

- Functions should do one thing only, not many
- Functions should rather be short

Practical Exercise

Practical Exercise

- Check out the sketch in code/rects_01
  - Read & understand the code, improve it where necessary
- Define a minimum size of 75 for the rectangles' width & height
- Ensure that all rectangles drawn lie inside the screen

Recap & Warm-up contd.

Array Data Structure

- In a second warm-up exercise, we will additionally use a more complex data structure — an array

```
int[] numbers = new int[3];
numbers[0] = 1;
numbers[1] = 2;
numbers[2] = 3;
```

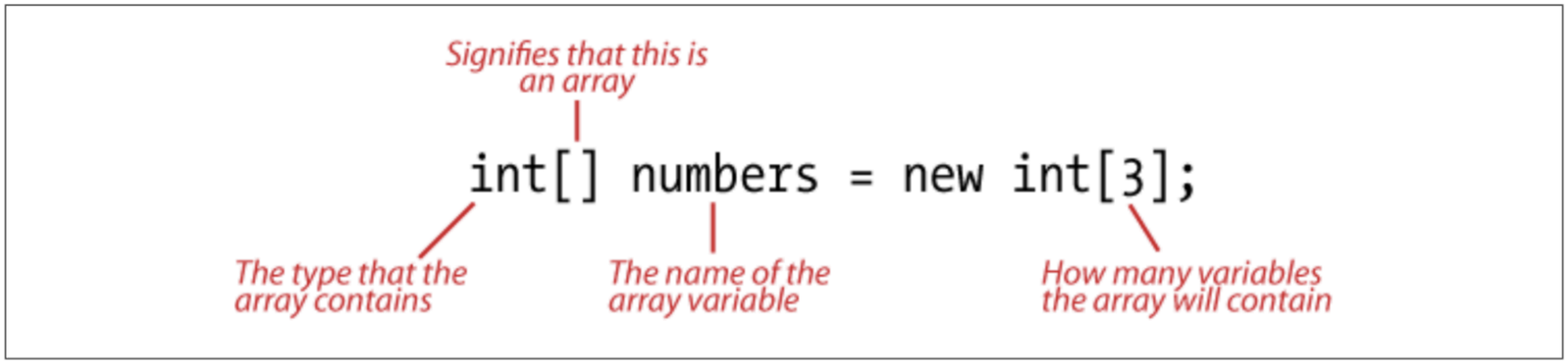


Image credit [JN09], p. 30.

Array Data Structure

- An array contains only elements of the same type, i.e., integer
- An array's size n defines the number of elements it can store
- An array uses an index i to access the value of the i'th element

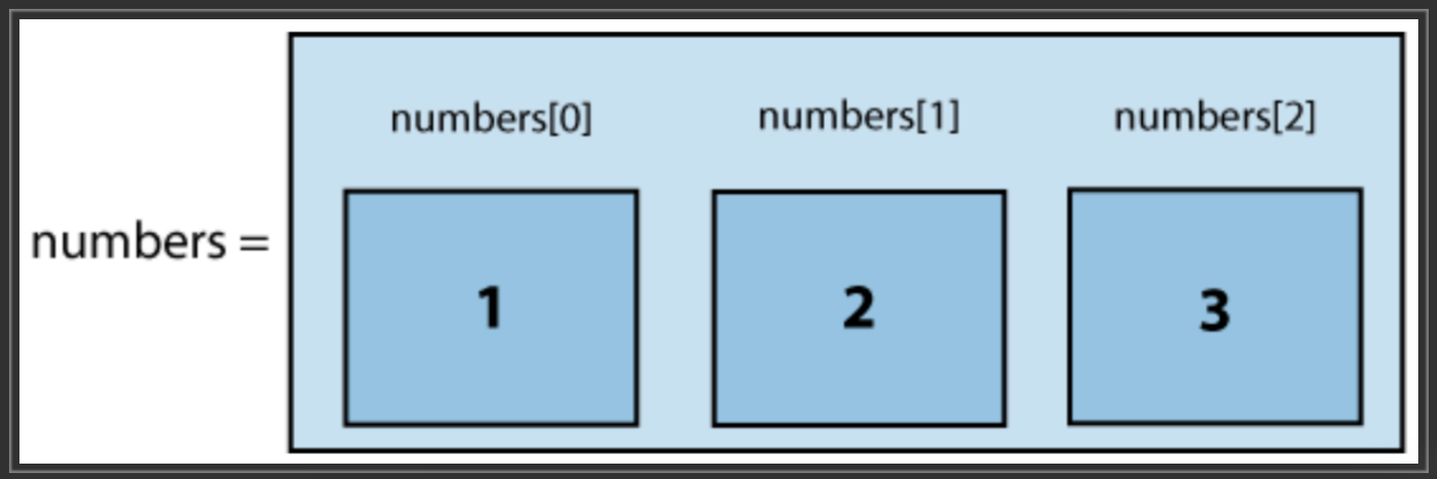


Image credit [JN09], p. 30.

```
int i = 1;
println( numbers[i] );
// results in „2“
```

Practical Exercise

Practical Exercise

- Now add interactivity to the sketch
  - Add color to the last rectangle drawn
    - a) when you move the mouse over the image, i.e., green
    - b) when the mouse is inside the last rectangle, i.e., red
      - How can this be implemented?

Practical Exercise

- Now add some interactivity to the first exercise
  - Add color to the last rectangle drawn
    - a) when you move the mouse over the image, i.e., green
    - b) when the mouse is inside the last rectangle, i.e., red
  - Store the last rectangle's parameters in an Array
  - When moving the mouse, set its' color to, e.g., green
  - When the mouse is in the rectangle, set its' color to, e.g., red

Object-Oriented Programming

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

Object-Oriented Programming

- Object-oriented programming (OOP) is a programming paradigm that is based on the idea of creating custom data structures, so called **objects**

- Objects are defined by their own variables & functions called **properties** & **methods**, respectively

Functions revisited

- So far, we coded in a way that is referred to as
  - **function-based** or **procedural**

- functions are used as the main building blocks
- functions work as units of reusable problem solvers
- functions help structure the code

Functions revisited

- Functions can be thought of as data processors
  - they (can) receive data as input
  - they do some kind of calculation
  - they (can) return data as output

```
15 boolean drawRectangle(float x, float y, float w, float h )
16 {
17   rect(x, y, w, h);
18   return true;
19 }
```

Functions revisited

- Functions can be thought of as data processors
  - they (can) receive data as input
  - they do some kind of calculation
  - they (can) return data as output

```
15 boolean drawRectangle(float x, float y, float w, float h )
16 {
17   rect(x, y, w, h);
18   return true;
19 }
```

- What if we want to specify more specific data structures and functions that operate on these data structures?

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

```
class myRect

    dimX
    dimY
```

mySmall
RectObject

50

50

myTall
RectObject

50

80

myBig
RectObject

100

70

Classes & Objects

- In its basic form a class requires a
  - class name
  - one or more constructors
  - properties
  - methods

```
83  class myRect
84  {
85    // properties
86    float positionX = 0;
87    float positionY = 0;
88    float dimensionX = 50;
89    float dimensionY = 50;
90
91    // constructors
92    myRect() {}
93
94    myRect(float theWidth, float theHeight)
95    {
96      dimensionX = theWidth;
97      dimensionY = theHeight;
98    }
99
100   myRect(float theX, float theY, float theWidth, float theHeight)
101   {
102     positionX = theX;
103     positionY = theY;
104     dimensionX = theWidth;
105     dimensionY = theHeight;
106   }
107
108   // methods
109   void draw() {
110     fill(0, 150, 0);
111     rect( positionX, positionY, dimensionX, dimensionY);
112   }
113
114   float getPositionX() { return positionX; }
115   float getPositionY() { return positionY; }
116   float getHeight() { return dimensionY; }
117   float getWidth() { return dimensionX; }
118
119   void setPositionX(float theX) { positionX = theX; }
120   void setPositionY(float theY) { positionY = theY; }
121   void setWidth(float theWidth) { dimensionX = theWidth; }
122   void setHeight(float theHeight) { dimensionY = theHeight; }
123 }
```

Classes & Objects

- Creating the default object of that class looks like this

```
 3 void setup()
 4 {
 5   size(600, 600);
 6   background(255);
 7
 8   myRect smallRect = new myRect();
 9   smallRect.draw();
10 }
```

```
 83 class myRect
 84 {
 85   // properties
 86   float positionX = 0;
 87   float positionY = 0;
 88   float dimensionX = 50;
 89   float dimensionY = 50;
 90
 91   // constructors
 92   myRect() {}
 93
 94   myRect(float theWidth, float theHeight)
 95   {
 96     dimensionX = theWidth;
 97     dimensionY = theHeight;
 98   }
 99
100   myRect(float theX, float theY, float theWidth, float theHeight)
101   {
102     positionX = theX;
103     positionY = theY;
104     dimensionX = theWidth;
105     dimensionY = theHeight;
106   }
107
108   // methods
109   void draw() {
110     fill(0, 150, 0);
111     rect( positionX, positionY, dimensionX, dimensionY);
112   }
113
114   float getPositionX() { return positionX; }
115   float getPositionY() { return positionY; }
116   float getHeight() { return dimensionY; }
117   float getWidth() { return dimensionX; }
118
119   void setPositionX(float theX) { positionX = theX; }
120   void setPositionY(float theY) { positionY = theY; }
121   void setWidth(float theWidth) { dimensionX = theWidth; }
122   void setHeight(float theHeight) { dimensionY = theHeight; }
123 }
```

Practical Exercise

Practical Exercise

- Now we will improve the processing sketch we have created so far with a custom rectangle class and corresponding objects

  - Check out code/rects_02 as a reference
  - Add a rectangle class to your sketch
    - Define a constructor
    - Define properties (aka „class member variables“)
    - Define methods (aka „class member functions“)
  - Re-write the sketch so that it uses your newly introduced class

  - Do you identify advantages and/or disadvantages?

Object-Oriented Programming contd.

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

OOP & System Design

- OOP is an abstraction mechanism
- to organize software components
- to group similar functionality into logical units / objects
- to define how the objects interact with each other

OOP & System Design

- In their simplest form, **objects** are **custom data structures**

- A **data structure** specifies how to store & organize data

- For instance, an Array is a **simple data structure**
  - it allows to store data of one specific data type
  - it organizes data in the form of a sequence

```
float[ ] degrees = new float[500];
for (int i=0; i < degrees.length; i++)
    degrees[i] = 25.0f;
```

OOP & System Design

- A **custom data structure** is a data structure that has been defined by **you**, the developer

- A custom data structure specifies how to store and organize data to solve a specific “custom” task

- Often, you would refer to a custom data structure simply as an **object**

OOP & System Design

- In terms of object-oriented programming, the notion of an object does not only refer to a custom data structure

- Objects can rather be understood as logical components of a (whole) system

OOP & System Design

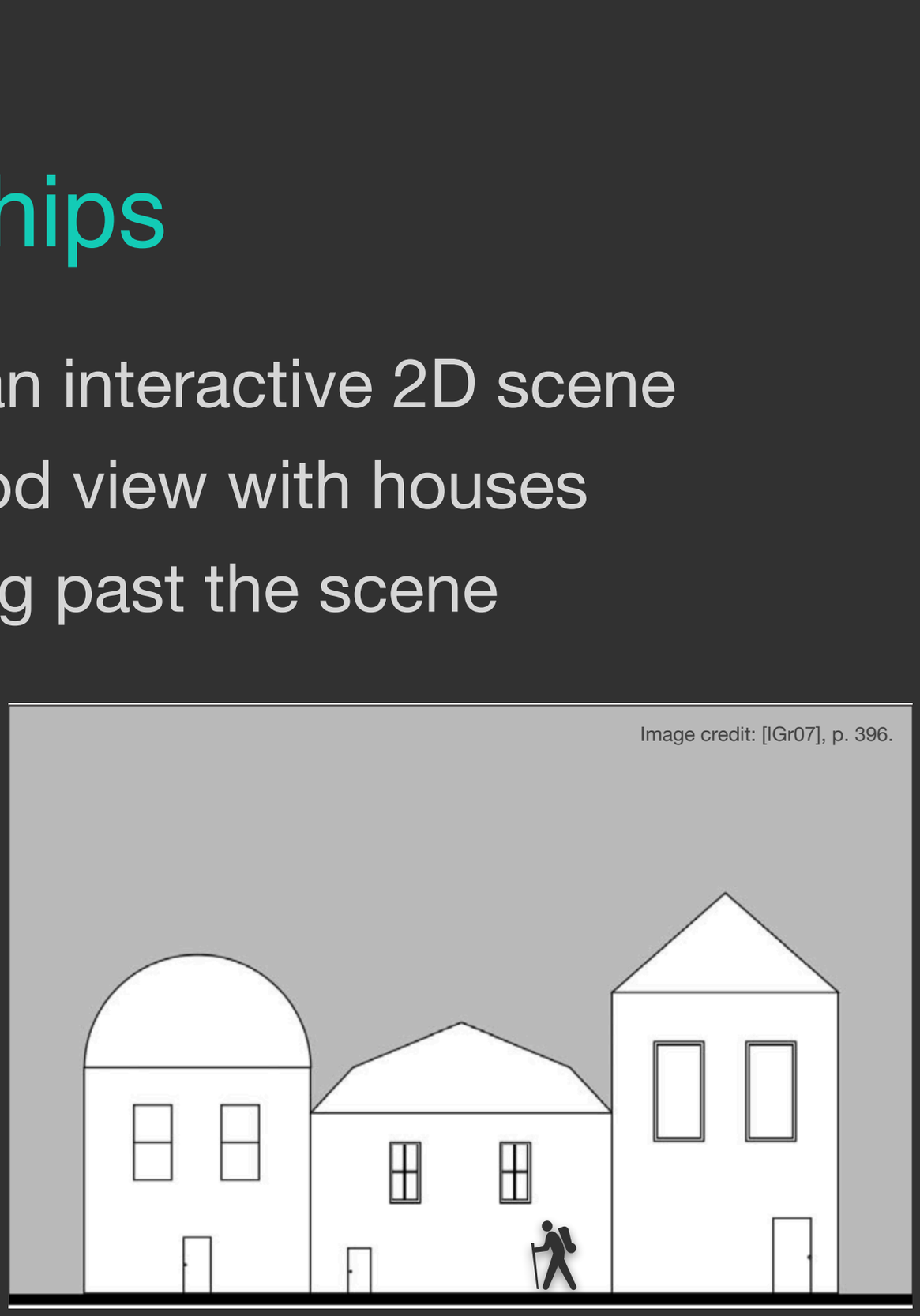
- In object-oriented programming you define
  - individual objects with properties & functionality (applicable in various contexts)
  - how the individual objects interact with each other and / or relate to each other

OOP & System Design

- In object-oriented programming you define
  - individual objects with properties & functionality (applicable in various contexts)
  - how the individual objects interact with each other and / or relate to each other
- Too abstract? Let’s look at a practical example!

Objects & Relationships

- Imagine you want to build an interactive 2D scene consisting of a neighborhood view with houses and a little character walking past the scene



Objects & Relationships

- Imagine you want to build an interactive 2D scene consisting of a neighborhood view with houses and a little character walking past the scene

- What would correspond to “the (whole) system”?

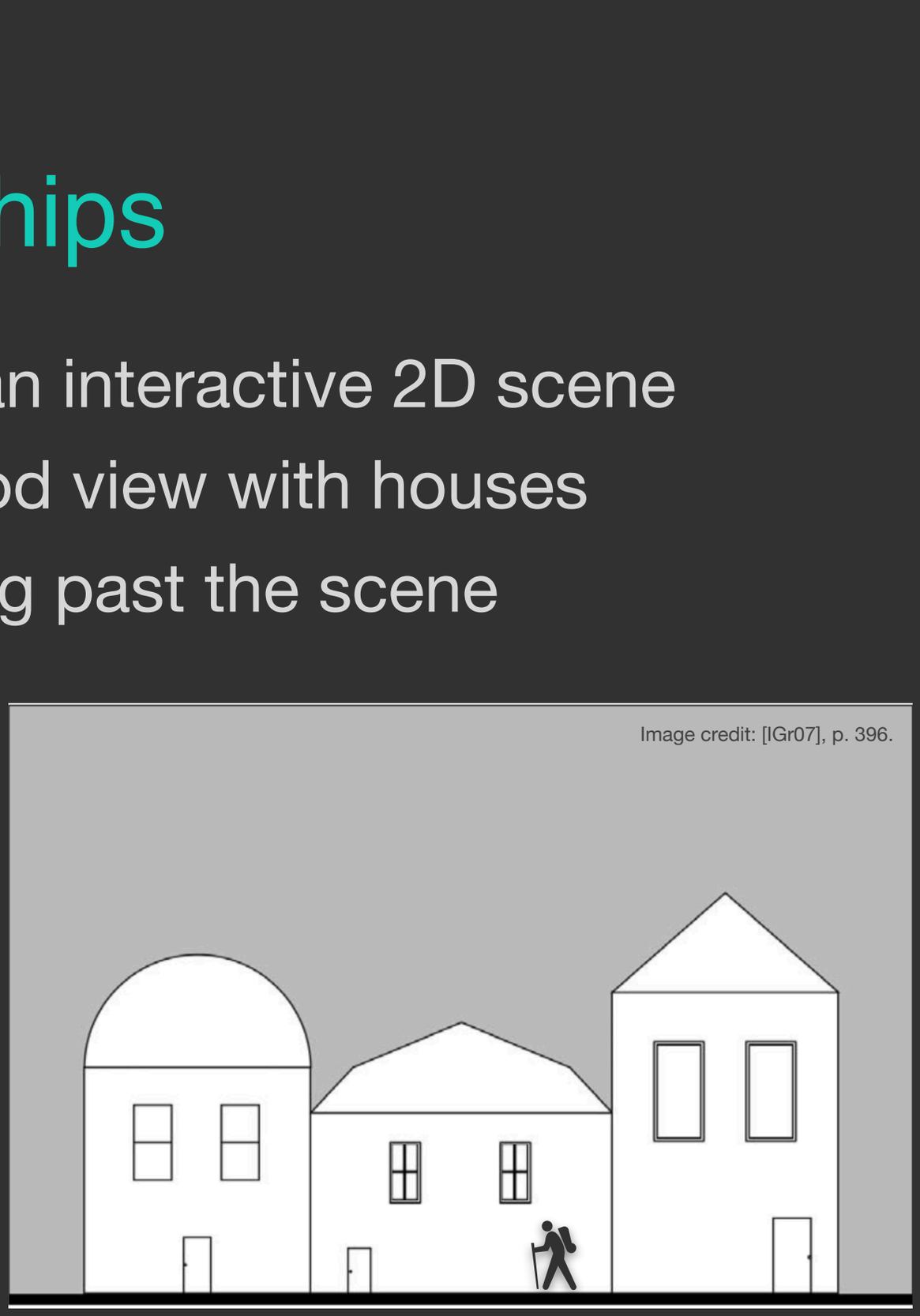


Image credit: [IGr07], p. 396.

Objects & Relationships

- Imagine you want to build an interactive 2D scene consisting of a neighborhood view with houses and a little character walking past the scene

- What would correspond to “the (whole) system”?
  - the 2D scene

Objects & Relationships

- Imagine you want to build an interactive 2D scene consisting of a neighborhood view with houses and a little character walking past the scene

- What would correspond to the “objects”?

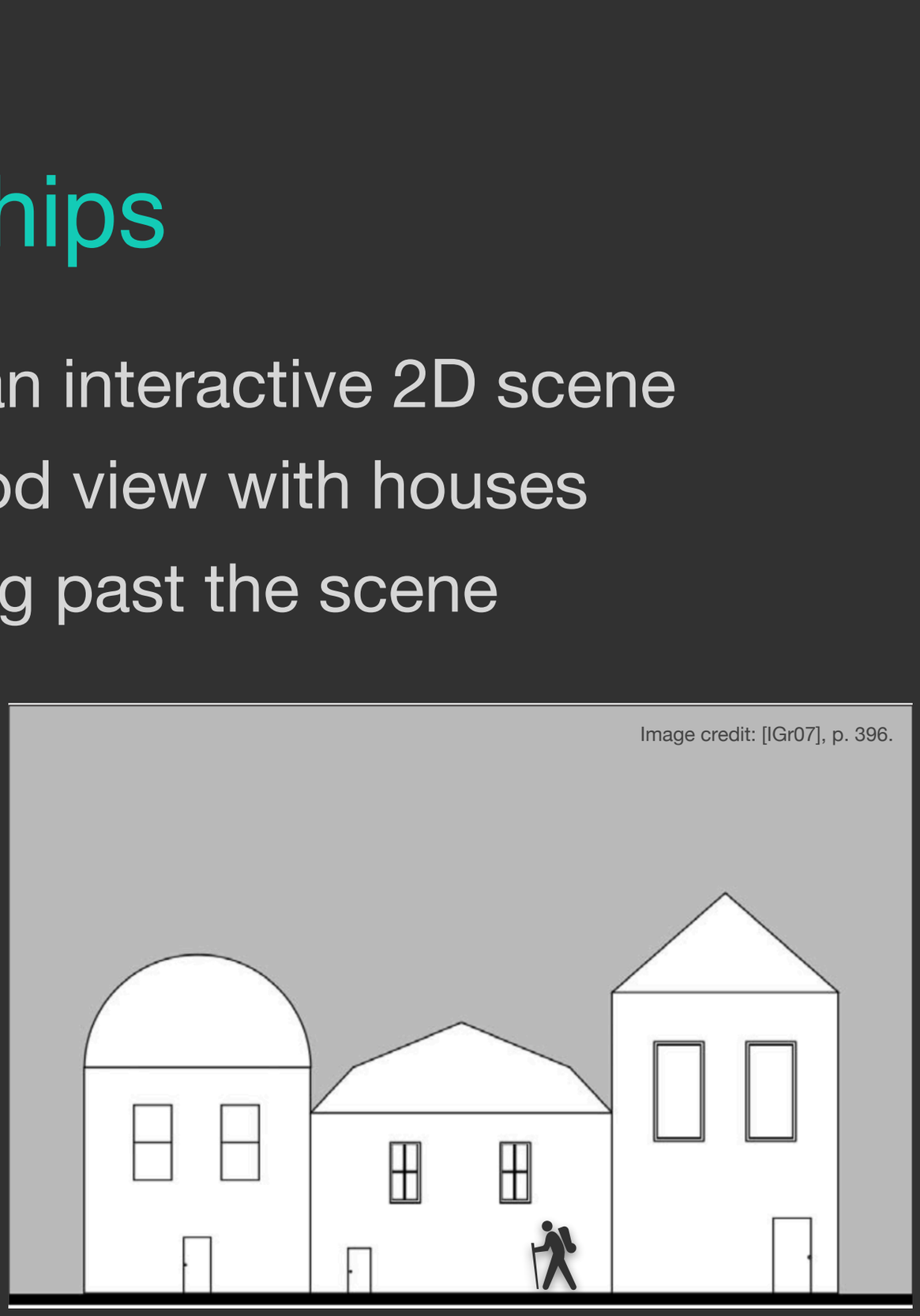


Image credit: [IGr07], p. 396.

Objects & Relationships

- Imagine you want to build an interactive 2D scene consisting of a neighborhood view with houses and a little character walking past the scene

- What would correspond to the “objects”?
  - houses
  - character
  - …?

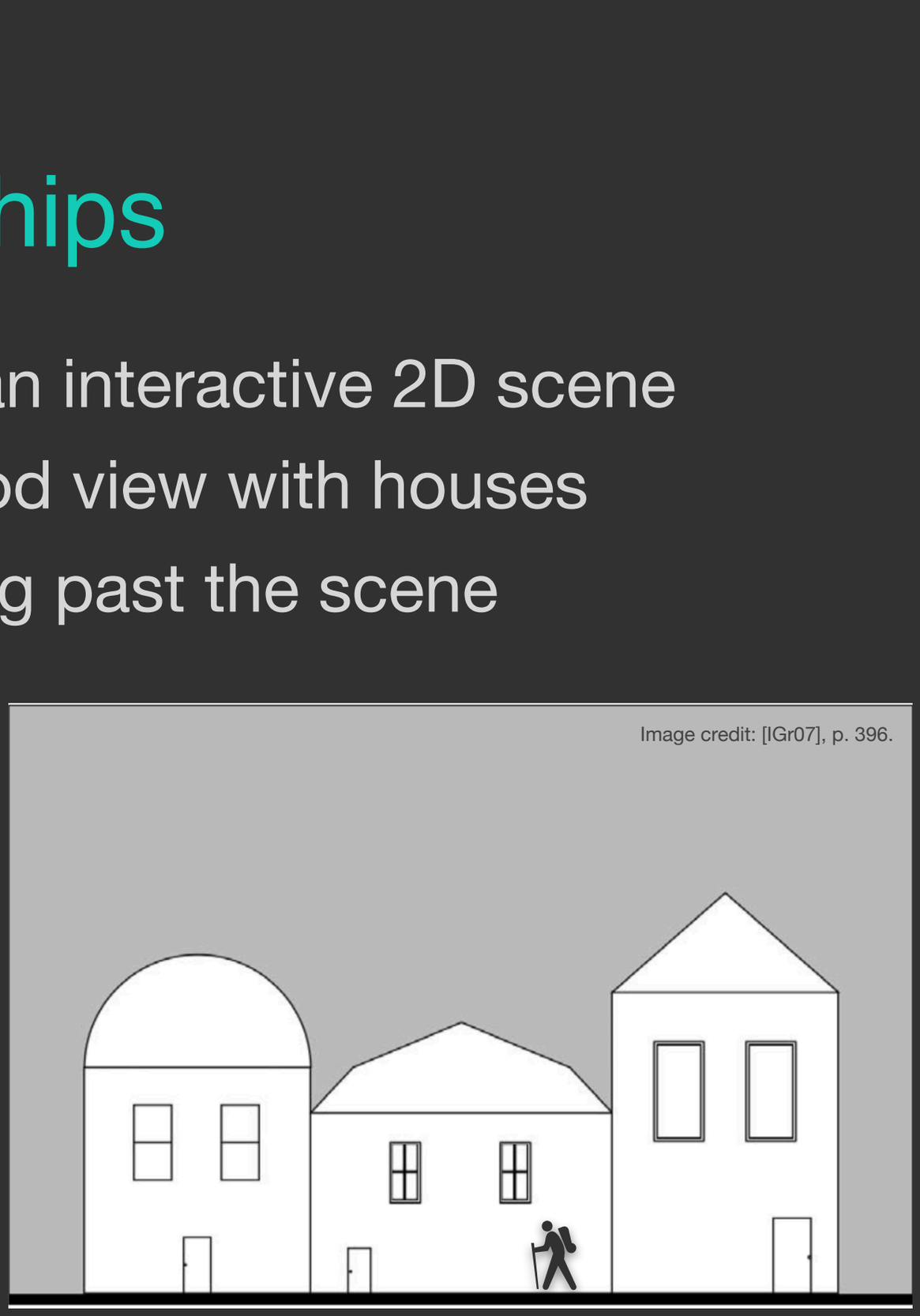


Image credit: [IGr07], p. 396.

Objects & Relationships

- In terms of **logical components / objects**, we define 3 houses and 1 character object

  - Each house object has specific properties like, i.e., windows, door, roof, size, …

- The character, too, has specific properties like, i.e., size, direction, …

Objects & Relationships

- In terms of **logical components / objects**, we define 3 houses and 1 character object
  - Each house object has specific properties like, i.e., windows, door, roof, size, …
    these are objects, too
- The character, too, has specific properties like, i.e., size, direction, …

Objects & Relationships

- In terms of **object functionality**
  - Each house, window, door, roof object has functionality that allows to position the object in the scene
  - The character, too, has functionality that allows to move them past the houses

Objects & Relationships

- In terms of **object relationships**
  - Each house object is composed of windows, doors, and roof objects
- This is a design aspect in OOP called “**Composition**”
  - objects can be composed of other objects
  - complex objects can be broken down into smaller ones simplifying many aspects of SW design

Objects & Relationships

- Individual objects with properties & functionality (applicable in various contexts)
  - houses, windows, doors, …
- How the individual objects interact with each other and / or relate to each other
  - functionality of houses, characters, …
  - composition of windows, doors, roofs to form a house

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 30 April, 10:00

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