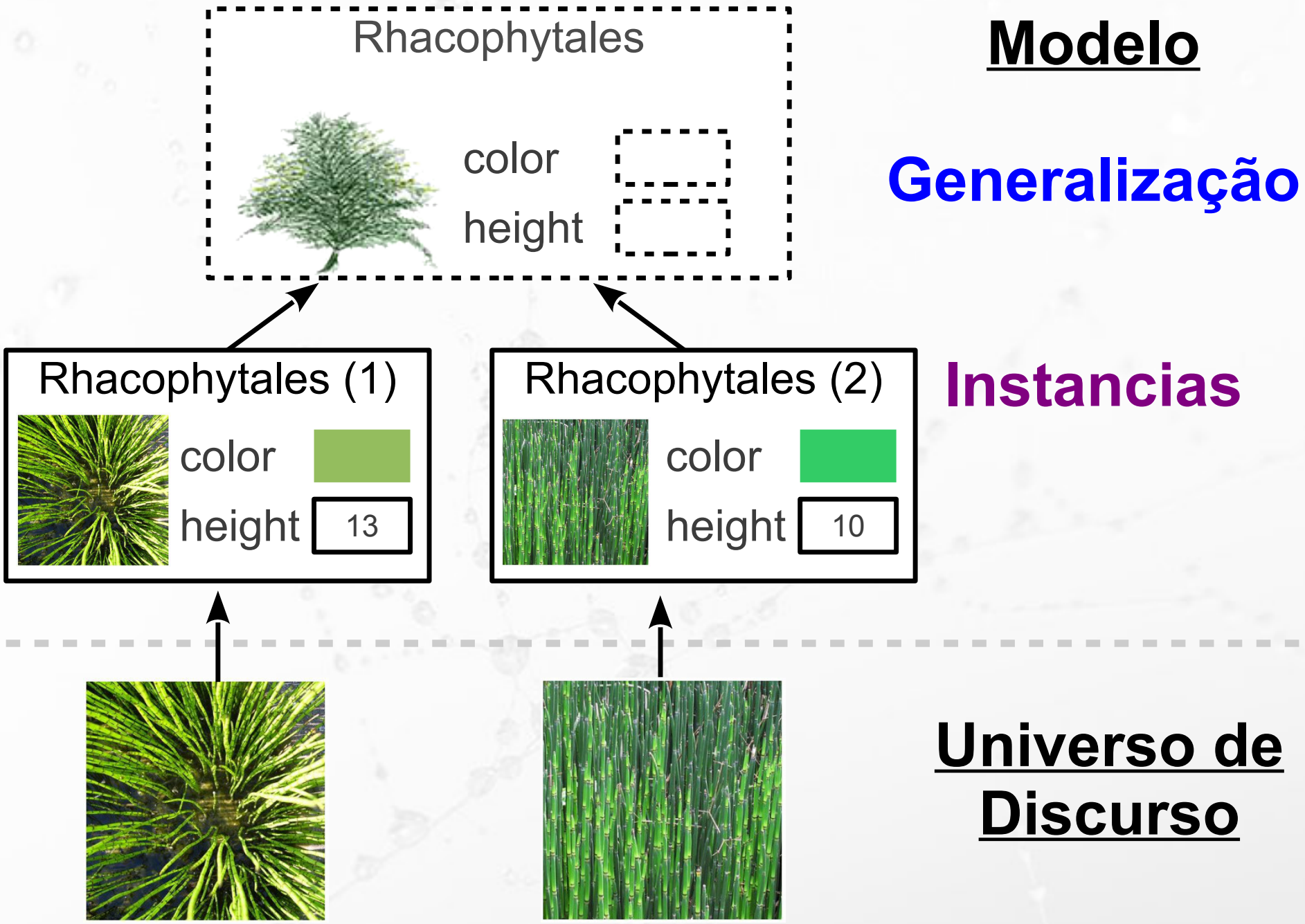
Programação Orientada a Objetos

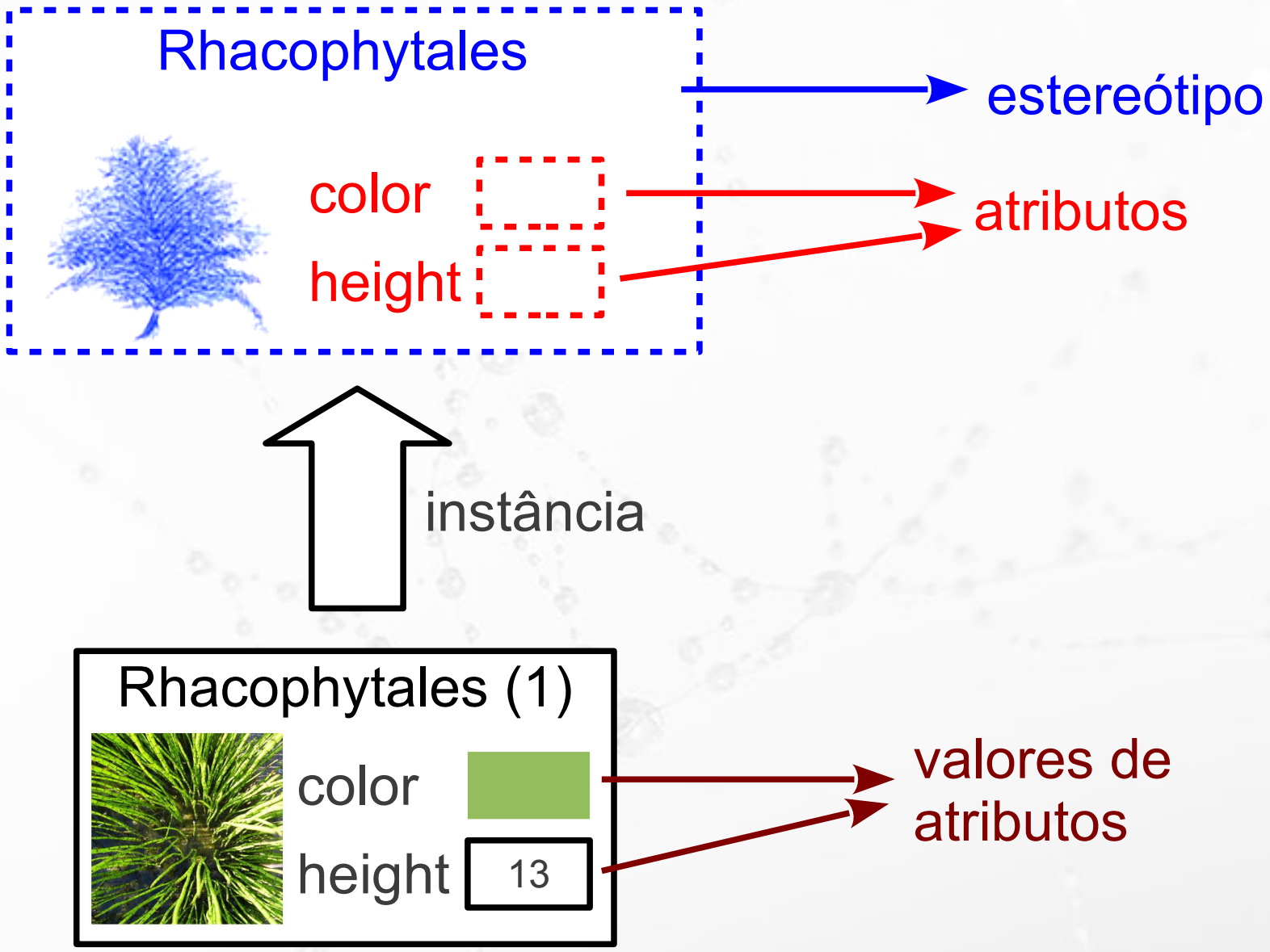
Formalismos e Programação OO
Objetos e Classes

André Santanchè e Oscar Rojas
Institute of Computing – UNICAMP
Março 2015

Formal

Estereótipos / Classes

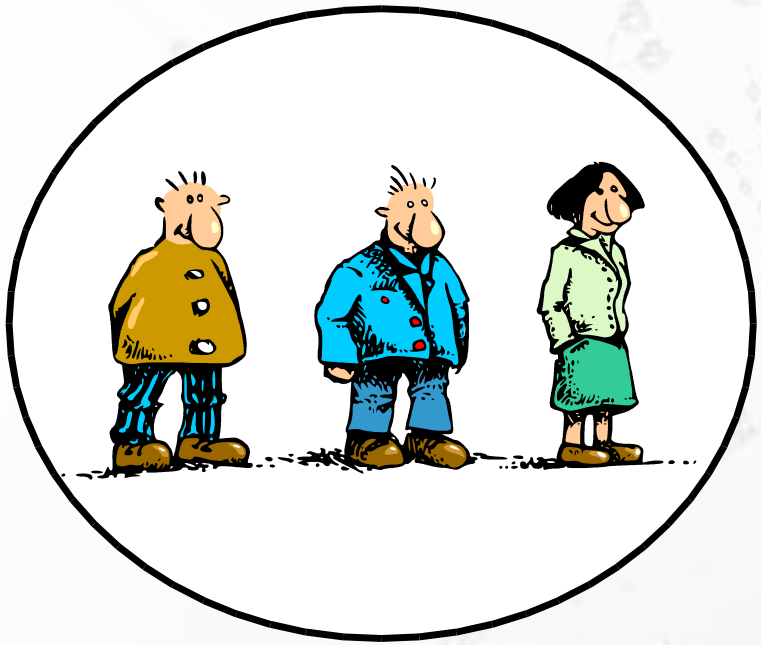

Rhacophytales
color
height
Rhacophytales (1)
color
height
13
Rhacophytales (2)
color
height
10
Modelo
Generalização
Instancias
Universo de Discurso


Rhacophytales
estereótipo
color
height
atributos
instância
Rhacophytales (1)
color
height
13
valores de
atributos

ER: Tipo Entidade

Tipo Entidade

- Tipo Entidade ou Conjunto de Entidades
  - conjunto não disjunto
  - entidades similares – mesmos atributos

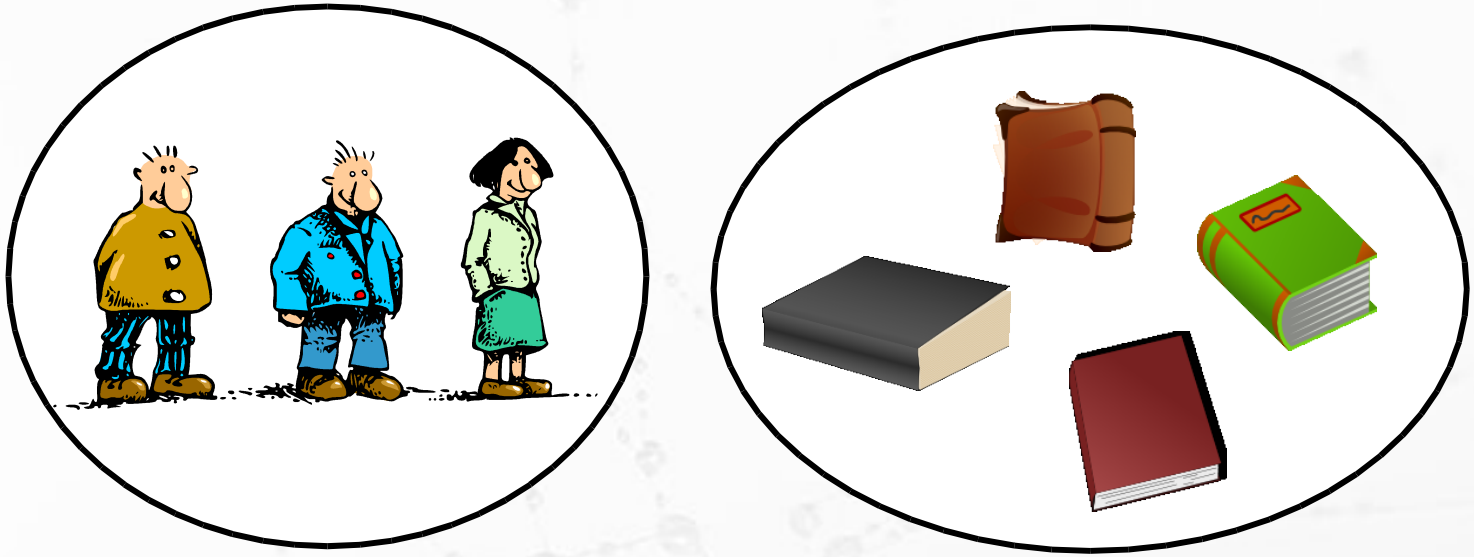


Conjunto
de Pessoas

Conjunto
de Livros

Tipo Entidade

- Representação:



Conjunto de Pessoas

Conjunto de Livros

Pessoa

Livro

OO: Classe

Abstrações em Computação

Tipo Abstrato de Dados

Tipo Abstrato de Dados (TAD)
Abstract Data Type (ADT)

- “O termo 'tipo abstrato de dados' se refere ao conceito matemático básico que define um tipo de dados” (Tenenbaum, 1990)
- Conceito matemático
  - Não considera aspectos de implementação
    - Ex.: eficiência de tempo e espaço

(Tenenbaum, 1990)

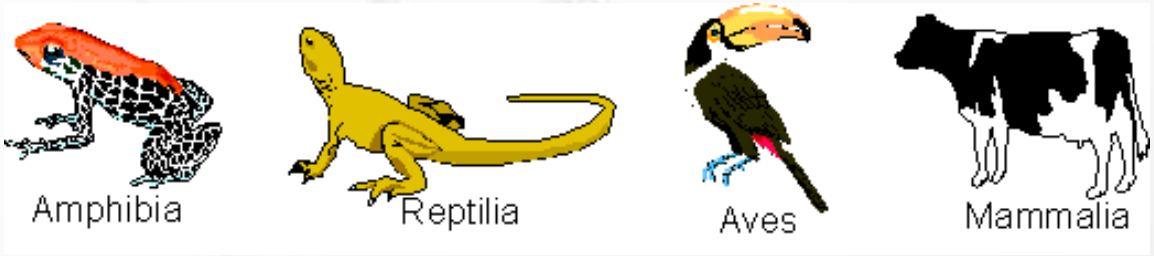
Tipo Abstrato de Dados (TAD)
Abstract Data Type (ADT)

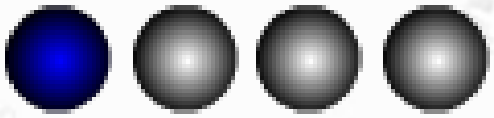
- “Um tipo abstrato de dados define uma classe de objetos abstratos que é completamente caracterizada pelas operações disponíveis nestes objetos. Isto significa que um tipo abstrato de dados pode ser definido pela definição e caracterização das operações daquele tipo.” (Liskov, 1974)

Classe

"Numa série ou num conjunto, grupo ou divisão que apresenta características ou atributos semelhantes." (Ferreira, 1989)

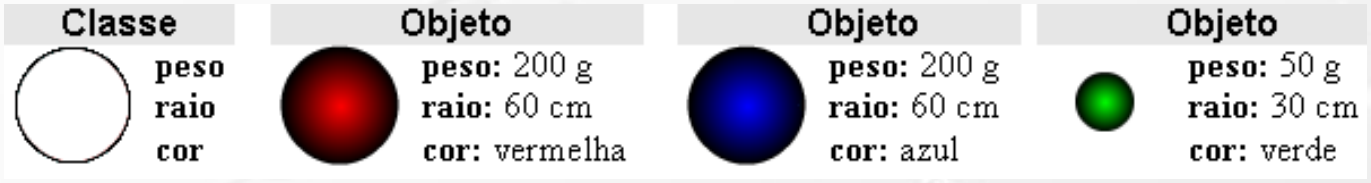
- Classificação de Carl Linné





Classe

- Quando realizamos uma classificação de objetos, identificamos o seu comportamento e as características que eles possuem em comum.
- Classes definem:
  - Atributos que irão descrever o objeto;
  - Métodos que definem o comportamento dos mesmos.



Objetos e Classes

- Os objetos são organizados/divididos em grupos chamados classes.

- Objetos da mesma classe têm:
  - o mesmo conjunto de atributos (os valores dos atributos podem ser diferentes);
  - o mesmo conjunto de métodos.

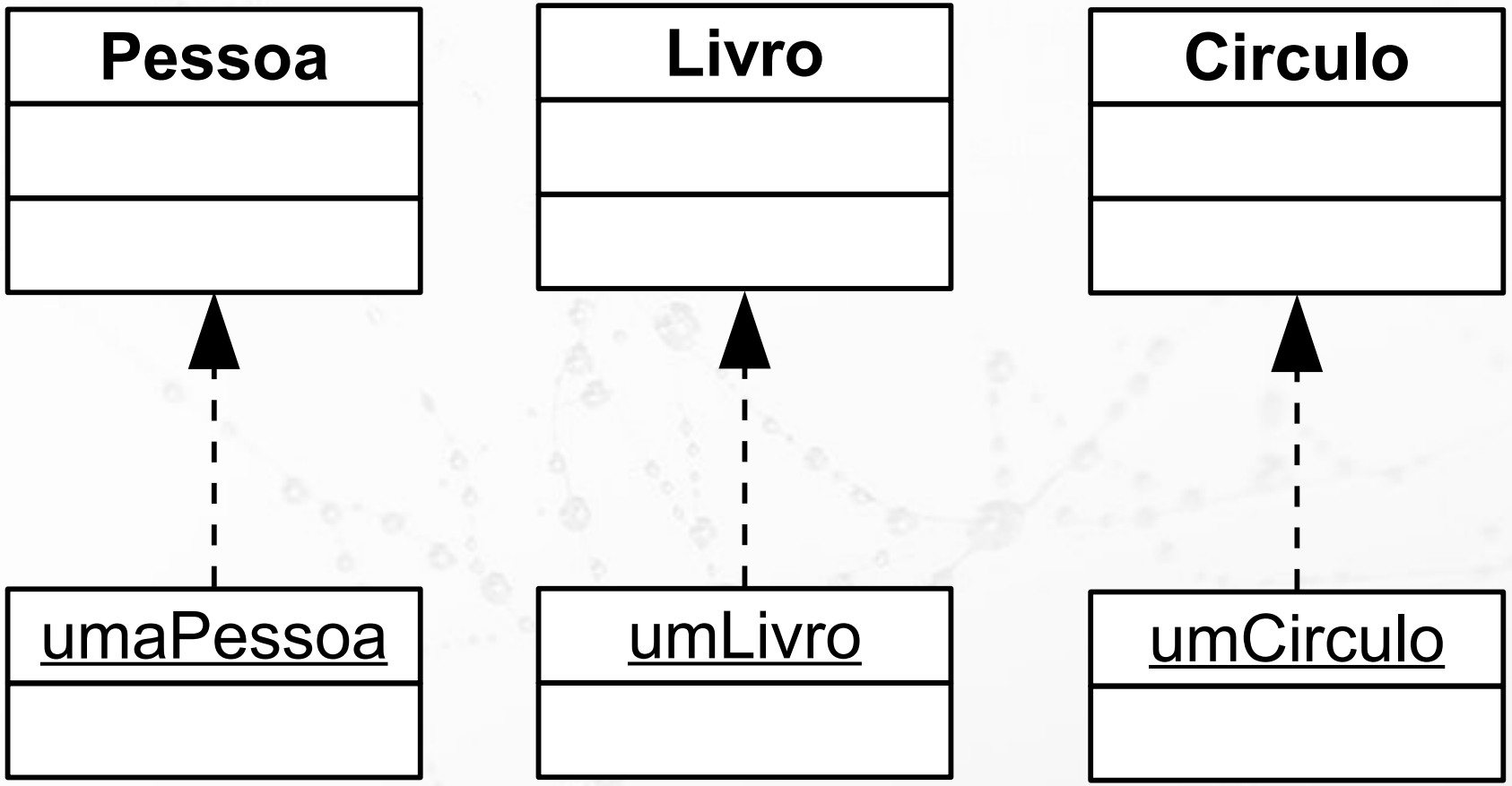
UML
Unified Modeling Language

- http://www.uml.org/
- Desenvolvida entre 1994-96
- Criadores
  - Grady Booch, Ivar Jacobson and James Rumbaugh na Rational Software
- Padrão OMG em 1997
  - OMG - Object Management Group
  - http://omg.org/

(Wikipedia, 2015)

UML
Classe

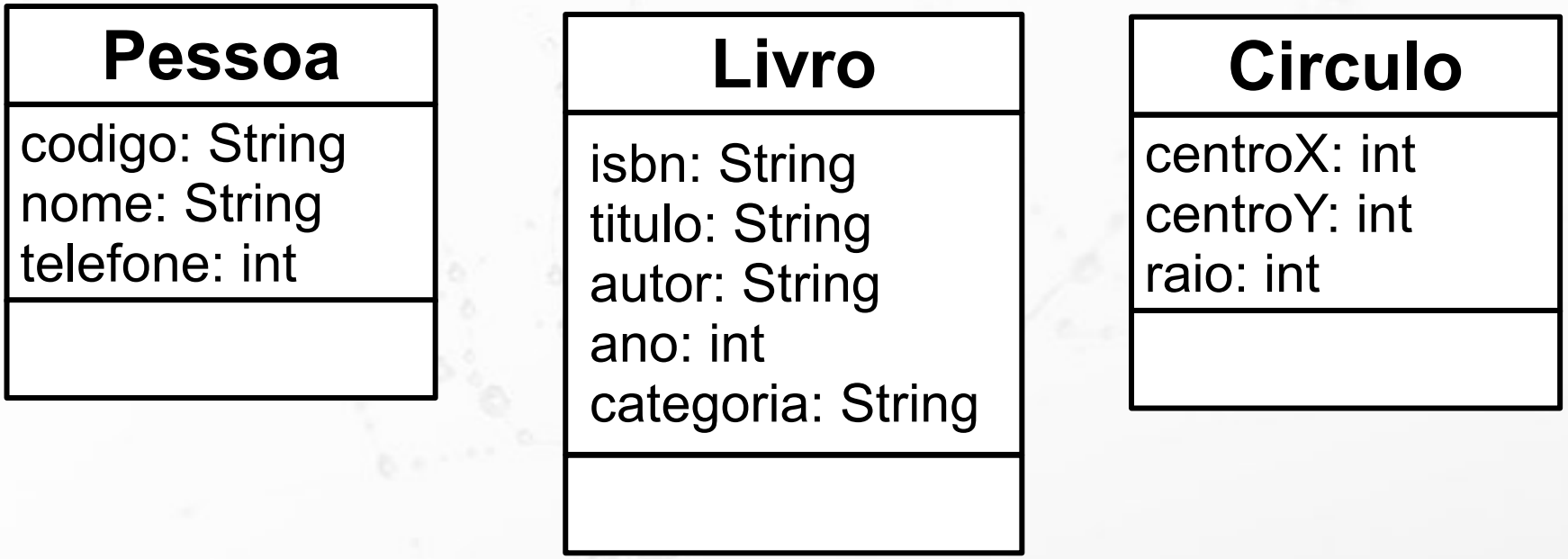
UML
Instância de Classe



UML

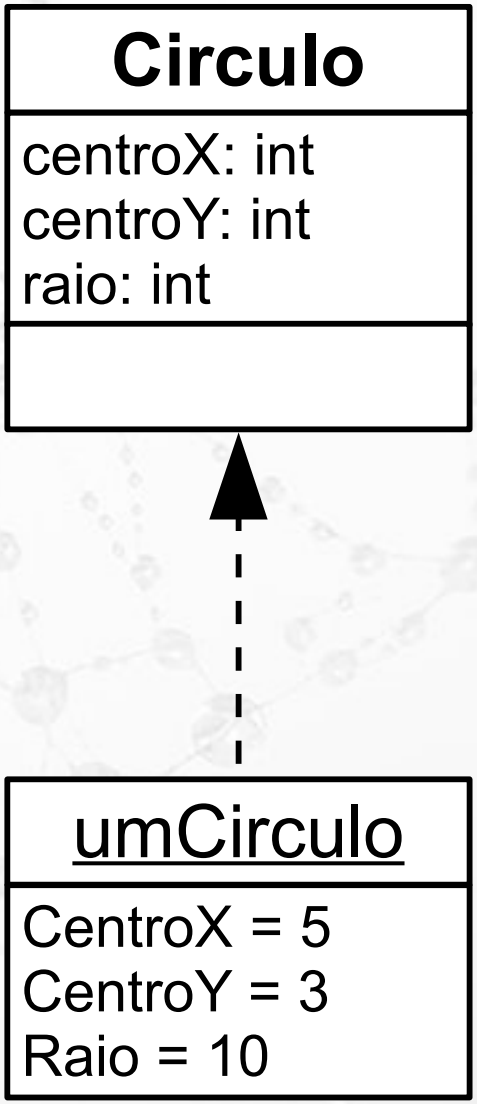
Instância de Classe (alternativa)

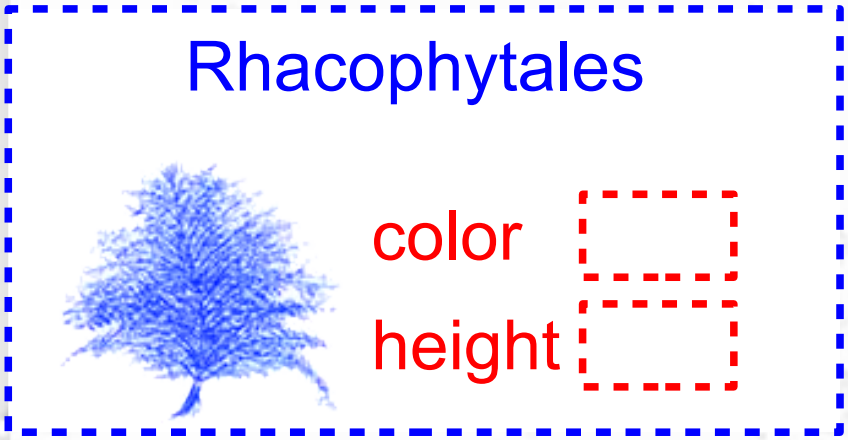
UML
Atributos (propriedades)

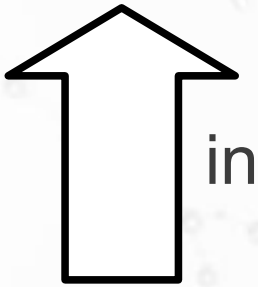


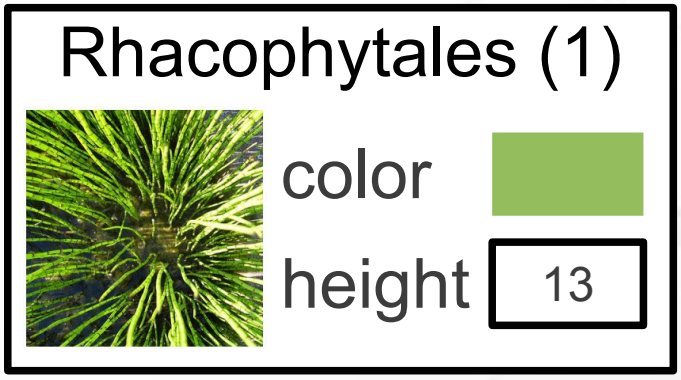
UML

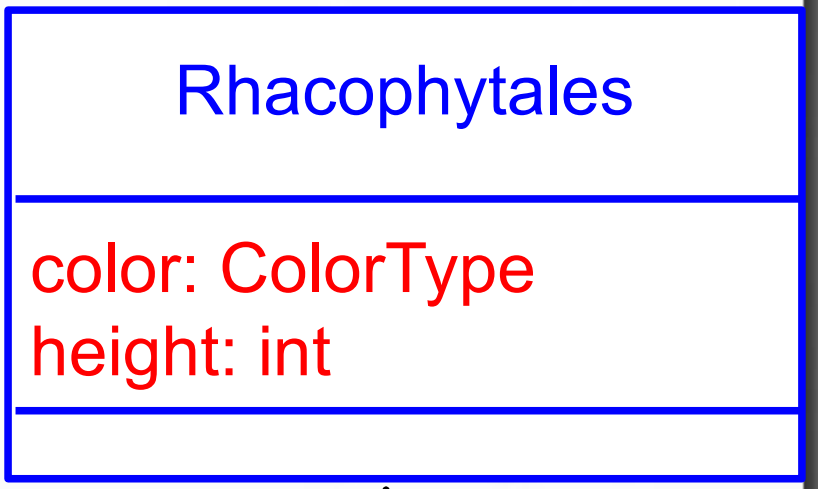
Instância com valores de atributos

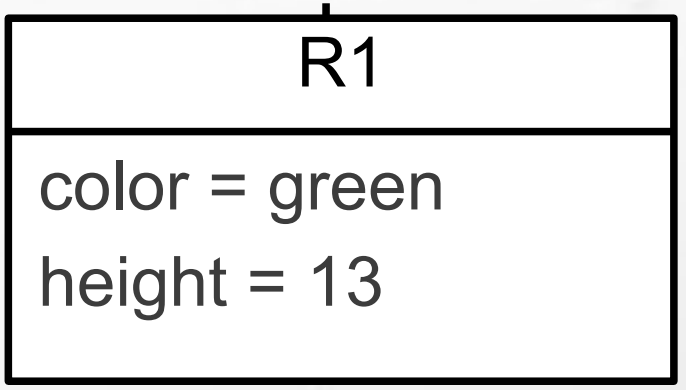



Rhacophytales
color
height

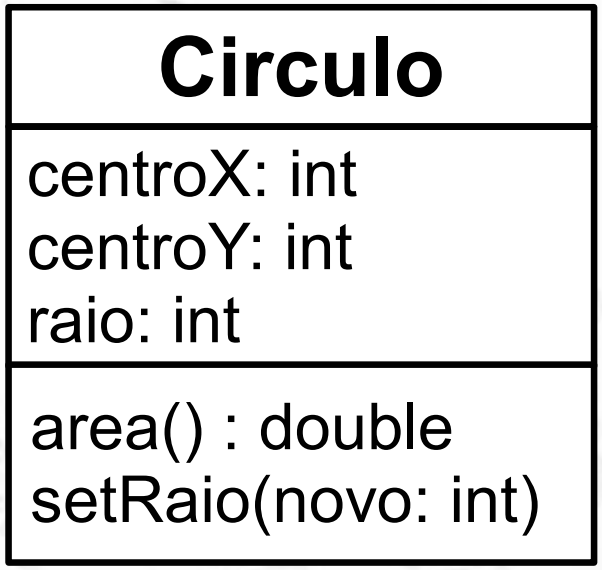

instância


Rhacophytales (1)
color
height
13

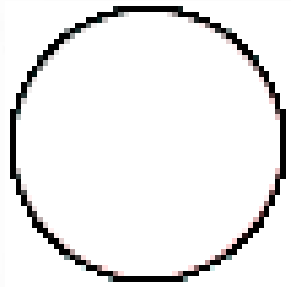

Rhacophytales
color: ColorType
height: int


R1
color = green
height = 13

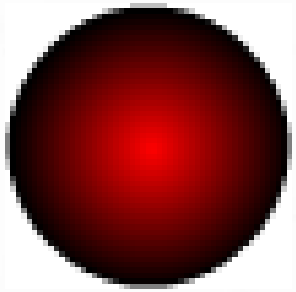
UML
Métodos (operações)



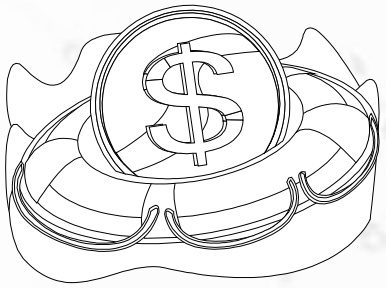
Exemplo de Classe
Esfera

Classe Esfera	
Atributos (nome, tipo)	
	(**peso**, real)
	(**raio**, real)
	(**elasticidade**, string)
	(**cor**, color)
Comportamento	
aumentar, diminuir, se mover	

Exemplo de Objeto
Esfera Vermelha

Objeto Esfera	
Atributos (nome, valor)	
	(**peso**, 200 g)
	(**raio**, 60 cm)
	(**elasticidade**, alta)
	(**cor**, vermelha)
Comportamento	
aumentar, diminuir, se mover	

Exemplo de Classe
Financiamento

Classe Financiamento	
Atributos (nome, tipo)	
	(**valor**, real)
	(**número de parcelas**, inteiro)
	(**percentual de juros**, real)
Comportamento	
calcula parcela	

Exemplo de Objeto
Um Financiamento

Objeto Financiamento	
Atributos (nome, valor)	
	(**valor**, R$ 150)
	(**número de parcelas**, 3)
	(**percentual de juros**, 1%)
Comportamento	
calcula parcela	

Segunda Face

Estrutura de Dados

Tipo Abstrato de Dados (TAD)
Abstract Data Type (ADT)

- “O termo 'tipo abstrato de dados' se refere ao conceito matemático básico que define um tipo de dados” (Tenenbaum, 1990)
- Conceito matemático
  - Não considera aspectos de implementação
    - Ex.: eficiência de tempo e espaço

(Tenenbaum, 1990)

Tipo Abstrato de Dados (TAD)
Abstract Data Type (ADT)

- “Um tipo abstrato de dados define uma classe de objetos abstratos que é completamente caracterizada pelas operações disponíveis nestes objetos. Isto significa que um tipo abstrato de dados pode ser definido pela definição e caracterização das operações daquele tipo.” (Liskov, 1974)

Classe – Arquitetura Modular

Estudo de Caso 1

Modularização Sucessiva

Estudo de Caso 1

Modularização Sucessiva

- Programa que calcula e apresenta o número de combinações possíveis que podem ser realizadas com bits, que variam de 1 a 8.
- Mostra etapas sucessivas do processo de modularização, até se chegar à classe.

Modularização Sucessiva
Primeira Versão

C

Programa sem modularização

Modularização Sucessiva
Primeira Versão

```c
#include <stdio.h>

int main()
{
    int combinacoes = 1,
        bits;

    for (bits = 1; bits <= 8; bits++)
    {
        combinacoes *= 2;
        printf("%d bits = %d combinacoes",
               bits, combinacoes);
    }

    return 0;
}
```

Modularização Sucessiva
Segunda Versão

C

Programa com modularização básica – apenas funções

Modularização Sucessiva
Segunda Versão

```
int combinacoes;

void inicializa() {
    combinacoes = 1;
}

int proximoNumeroCombinacoes() {
    combinacoes *= 2;
    return combinacoes;
}

int main() {
    int bits;
    inicializa();
    for (bits = 1; bits <= 8; bits++)
        printf("%d bits = %d combinacoes",
                bits, proximoNumeroCombinacoes());
    return 0;
}
```

Modularização Sucessiva
Segunda Versão

- Vantagens:
  - Módulos encapsulam a lógica do cálculo de combinações, de forma que ela possa ser reusada.

Modularização Sucessiva
Segunda Versão

- Problemas:
  - Os módulos dependem do programa principal que mantém a variável "combinacoes".
  - O programa principal se torna responsável por detalhes de implementação dos módulos, o que prejudica o reuso:
    - cada vez que um programa reusar os módulos precisara declarar a variável "combinacoes";
    - a nova variável "combinacoes" declarada pode entrar em conflito com uma já existente, o que exige modificação do código

Modularização Sucessiva
Terceira Versão

C

Tentativa de transferir a variável "combinacoes" para os módulos, a fim de remover a dependência

Modularização Sucessiva
Terceira Versão

```
void inicializa() {
    int combinacoes;
    combinacoes = 1;
}

int proximoNumeroCombinacoes() {
    int combinacoes;
    combinacoes *= 2;
    return combinacoes;
}

int main() {
    int bits;
    inicializa();
    for (bits = 1; bits <= 8; bits++)
        printf("%d bits = %d combinacoes",
                bits, proximoNumeroCombinacoes());
}
```

Modularização Sucessiva
Terceira Versão

- Erro de execução:
  - A variável local e criada e destruída a cada entrada/saída de cada um dos módulos, impossibilitando a continuidade desejada.

Modularização Sucessiva
Quarta Versão

C

Uma variável global é declarada e passada como parâmetro para os módulos

Modularização Sucessiva
Quarta Versão

```
void inicializa(int *combinacoes) {
    *combinacoes = 1;
}

int proximoNumeroCombinacoes(int *combinacoes) {
    *combinacoes *= 2;
    return *combinacoes;
}

int main() {
    int combinacoes;
    int bits;

    inicializa(&combinacoes);

    for (bits = 1; bits <= 8; bits++)
        printf("%d bits = %d combinacoes\n", bits,
                    proximoNumeroCombinacoes(&combinacoes));
}
```

Modularização Sucessiva
Quarta Versão

- Vantagens:
  - A variável do programa principal se torna independente da variável dos módulos (o nome pode ser diferente).
- Problemas:
  - O programa principal continua precisando declarar e manter a variável "combinacoes", o que ainda causa dependência dos módulos
  - Neste ponto esgotam-se as possibilidades da modularização baseada em procedures e functions.

Modularização Sucessiva

Quinta Versão

C

Os módulos menores (funções) são colocados dentro de um módulo maior

Modularização Sucessiva

Quinta Versão - bits05module.h

```
void inicializa();

int proximoNumeroCombinacoes();
```

Modularização Sucessiva
Quinta Versão - bits05module.c

```c
#include "bits05module.h"

static int combinacoes;

void inicializa()
{
    combinacoes = 1;
}

int proximoNumeroCombinacoes()
{
    combinacoes *= 2;
    return combinacoes;
}
```

Modularização Sucessiva
Quinta Versão - bits05.c

```c
#include "bits05module.h"

int main()
{
    int bits;

    inicializa();

    for (bits = 1; bits <= 8; bits++)
        printf("%d bits = %d combinacoes",
               bits, proximoNumeroCombinacoes());

    return 0;
}
```

Modularização Sucessiva
Quinta Versão

- Vantagens:
  - O módulo expõe apenas as interfaces das funções, escondendo detalhes da implementação
  - O módulo controla e mantém o estado da variável "combinacoes", que não e visível para o programa principal.

Modularização Sucessiva
Quinta Versão

- Problemas:
  - O módulo funciona para apenas uma instância. Se precisássemos de dois cálculos de combinações em paralelo teríamos problemas.
  - O uso de múltiplas instâncias é possível mas complicado.

Modularização Sucessiva
Sexta/Sétima Versão

C++
Classe

Modularização Sucessiva

Sétima Versão - Bits07Instancia.h

```
class Bits07Instancia
{
    int combinacoes;
    public:
        Bits07Instancia();
        int proximoNumeroCombinacoes();
};
```

Modularização Sucessiva

Sétima Versão - Bits07Instancia.cpp

```
#include "Bits07Instancia.h"


Bits07Instancia::Bits07Instancia()
{
    combinacoes = 1;
}


int Bits07Instancia::proximoNumeroCombinacoes()
{
    combinacoes *= 2;
    return combinacoes;
}
```

Modularização Sucessiva
Sétima Versão - Bits07.c

```
#include <stdio.h>


#include "Bits07Instancia.h"


int main () {
  Bits07Instancia objeto;


  int bits;
  for (bits = 1; bits <= 8; bits++)
        printf("%d bits = %d combinacoes\n", bits,
  objeto.proximoNumeroCombinacoes());
}
```

Modularização Sucessiva
Sexta Versão

Java

O módulo é transformada em uma classe

Modularização Sucessiva
Sexta Versão - Classe

```
package pt.c02oo.s01estudocaso.s06classe;

public class Bits06Classe
{
    static int combinacoes;

    static void inicializa()
    {
        combinacoes = 1;
    }

    static int proximoNumeroCombinacoes()
    {
        combinacoes *= 2;
        return combinacoes;
    }
}
```

Modularização Sucessiva

Sexta Versão – Programa Principal

```
package pt.c02oo.s01estudocaso.s06classe;

public class Bits06
{
    public static void main(String args[])
    {
        Bits06Classe.inicializa();
        for (int bits = 1; bits <= 8; bits++)
            System.out.println(bits + " = " +
                               Bits06Classe.proximoNumeroCombinacoes());
    }
}
```

Modularização Sucessiva
Sétima Versão

Java
Classe com instância

Modularização Sucessiva
Sétima Versão - Classe

```
package pt.c02oo.s01estudocaso.s07objeto;

public class Bits07Instancia
{
    int combinacoes;

    Bits07Instancia()
    {
        combinacoes = 1;
    }

    int proximoNumeroCombinacoes()
    {
        combinacoes *= 2;
        return combinacoes;
    }
}
```

Modularização Sucessiva
Sétima Versão - Programa Principal

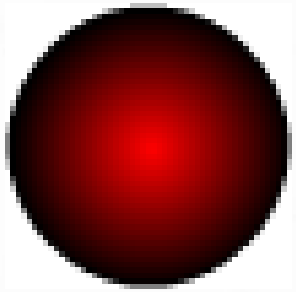
```
package pt.c02oo.s01estudocaso.s07objeto;

public class Bits07
{
    public static void main(String args[])
    {
        Bits07Instancia objeto;
        objeto = new Bits07Instancia();

        for (int bits = 1; bits <= 8; bits++)
            System.out.println(bits + " = " +
                               objeto.proximoNumeroCombinacoes());
    }
}
```

Objeto

"O que se apresenta à percepção com um caráter fixo e estável". (Ferreira, 1989)

- Objetos são caracterizados por:
  - identidade;
  - atributos;
  - comportamento.

Exemplo de Objeto
Esfera Vermelha

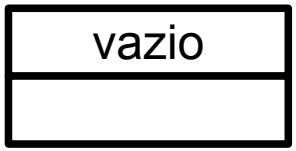
Objeto Esfera	
Atributos (nome, valor)	
	(**peso**, 200 g)
	(**raio**, 60 cm)
	(**elasticidade**, alta)
	(**cor**, vermelha)
Comportamento	
aumentar, diminuir, se mover	

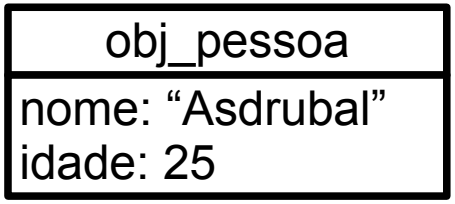
Exemplo de Objeto
Um Financiamento

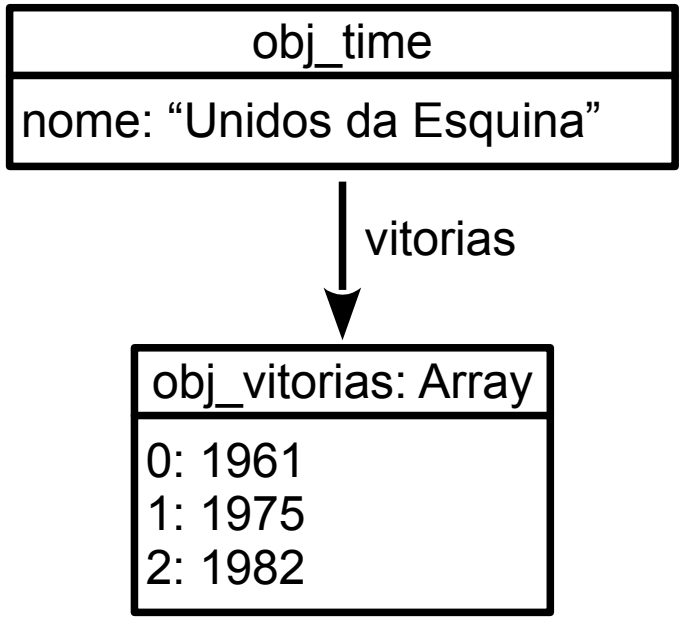
Objeto Financiamento	
Atributos (nome, valor)	
	(**valor**, R$ 150)
	(**número de parcelas**, 3)
	(**percentual de juros**, 1%)
Comportamento	
calcula parcela	

Objetos em JavaScript

Objetos JS

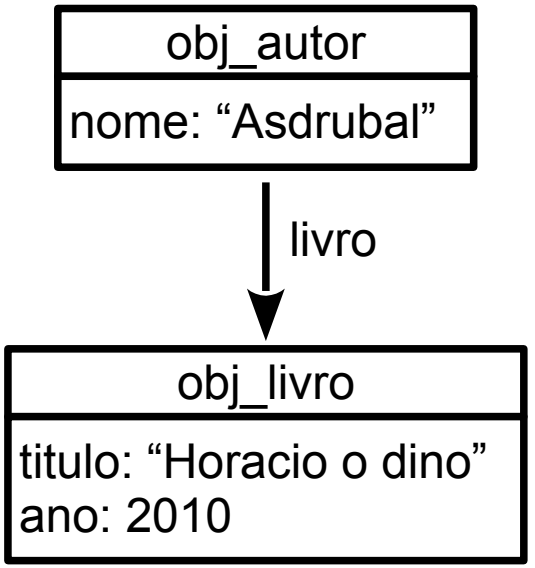
`{ }`	

`{`
`"nome": "Asdrubal",`
`"idade": 25`
`}`
	

`{`
`"nome": "Unidos da Esquina",`
`"vitorias": [1961, 1975, 1982]`
`}`
	

Objetos JS

```
{
 "nome": "Asdrubal",
 "livro": {
   "titulo": "Horacio o dino",
   "ano": 2010
  }
}
```



JavaScript

Objeto com Atributos (1)

```
function exemploObjetoAtributos1() {
  var circulo = {
    centroX : 5,
    centroY : 3,
    raio : 10
  };

  alert("Circulo: centro(" + circulo.centroX + ", " +
        circulo.centroY + "), raio " + circulo.raio);
}
```

JavaScript

Objeto com Atributos (2)

```
function exemploObjetoAtributos2() {
  var circulo = {};

  circulo.centroX = 5;
  circulo.centroY = 3;
  circulo.raio = 10;

  alert("Circulo: centro(" + circulo.centroX + ", " +
circulo.centroY + "), raio " + circulo.raio);
}
```

JavaScript

Objeto com Método (1)

```
function exemploObjetoMetodo1() {
  var circulo = {
     centroX : 5,
     centroY : 3,
     raio : 10,
     area : function() {
        return 3.1416 * this.raio * this.raio;
     }
  };

  alert("Circulo: centro(" + circulo.centroX + ", " +
        circulo.centroY + "), raio " + circulo.raio + ",
        area " + circulo.area());
}
```

JavaScript

Objeto com Método (2)

```
function exemploObjetoMetodo3() {
  var circulo = {};

  circulo.centroX = 5;
  circulo.centroY = 3;
  circulo.raio = 10;
  circulo.area = function() {
            return 3.1416 * this.raio * this.raio;
              }

  alert("Circulo: centro(" + circulo.centroX + ", " +
        circulo.centroY + "), raio " + circulo.raio +
        ", area " + circulo.area());
}
```

JSON
JavaScript Object Notation

- Padrão aberto de intercâmbio de objetos
- Baseado na notação JavaScript
- Incorporado ao ECMAScript (Ecma, 2011)
- Adotado por diversas linguagens (http://json.org/)

Stringify

- Serializando

```
var pessoa = {
  "nome": "Asdrubal",
  "idade": 25
};
var pessoaStr = JSON.stringify(pessoa);
```

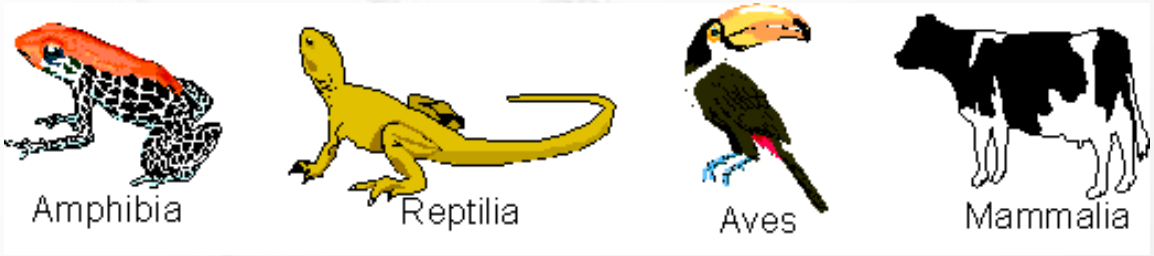
- Desserializando

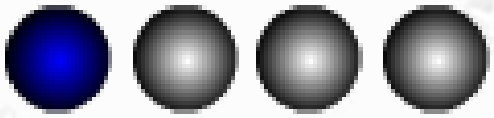
```
var pessoa2 = JSON.parse(pessoaStr);
```

Classe

"Numa série ou num conjunto, grupo ou divisão que apresenta características ou atributos semelhantes." (Ferreira, 1989)

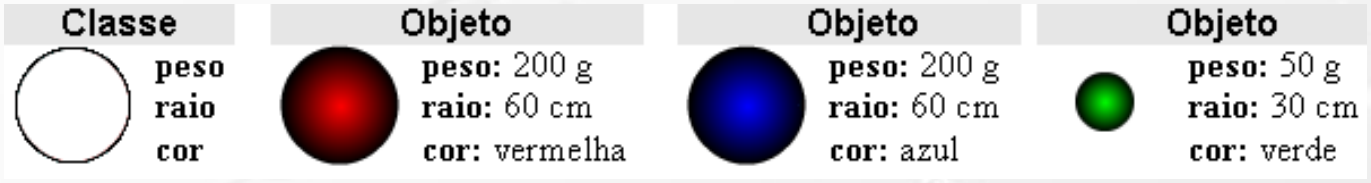
- Classificação de Carl Linné





Classe

- Quando realizamos uma classificação de objetos, identificamos o seu comportamento e as características que eles possuem em comum.
- Classes definem:
  - Atributos que irão descrever o objeto;
  - Métodos que definem o comportamento dos mesmos.

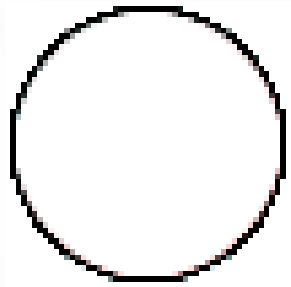


Objetos e Classes

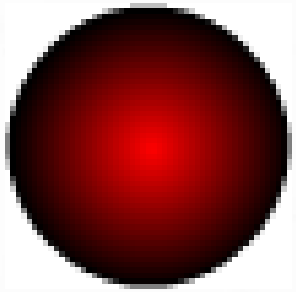
- Os objetos são organizados/divididos em grupos chamados classes.

- Objetos da mesma classe têm:
  - o mesmo conjunto de atributos (os valores dos atributos podem ser diferentes);
  - o mesmo conjunto de métodos.

Exemplo de Classe
Esfera

Classe Esfera	
Atributos (nome, tipo)	
	(**peso**, real)
	(**raio**, real)
	(**elasticidade**, string)
	(**cor**, color)
Comportamento	
aumentar, diminuir, se mover	

Exemplo de Objeto
Esfera Vermelha

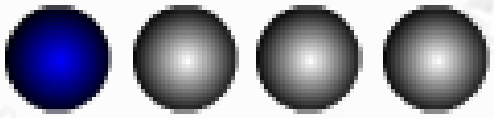
Objeto Esfera	
Atributos (nome, valor)	
	(**peso**, 200 g)
	(**raio**, 60 cm)
	(**elasticidade**, alta)
	(**cor**, vermelha)
Comportamento	
aumentar, diminuir, se mover	

Exemplo de Classe
Financiamento

Classe Financiamento	
Atributos (nome, tipo)	
	(**valor**, real)
	(**número de parcelas**, inteiro)
	(**percentual de juros**, real)
Comportamento	
calcula parcela	

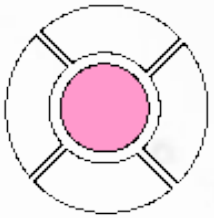
Exemplo de Objeto
Um Financiamento

Objeto Financiamento	
Atributos (nome, valor)	
	(**valor**, R$ 150)
	(**número de parcelas**, 3)
	(**percentual de juros**, 1%)
Comportamento	
calcula parcela	

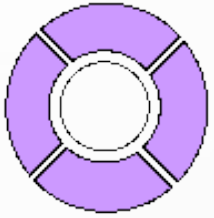


Classe

- Em Programação Orientada ao Objeto:



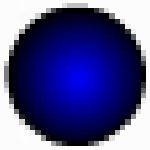
Atributos: dados que pertencem a cada instância da classe (objeto); são definidos sob a forma de variáveis.



Métodos: definem o comportamento do objeto; representados por módulos.

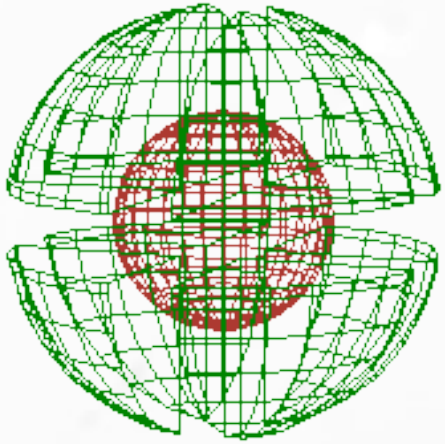
Objeto instância de Classe

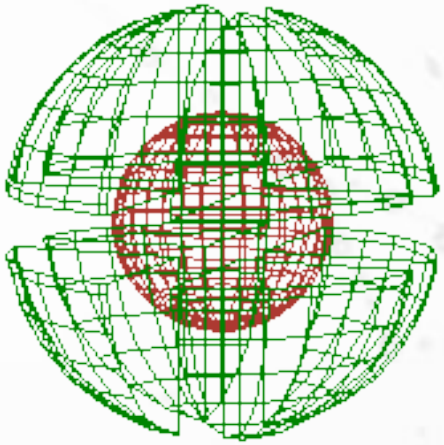
- Um Objeto consiste em uma instância de uma Classe
- A instância define:
  - identidade única
  - estado (representado pelos valores de seus atributos).



Objeto

A classe pode ser importada de uma biblioteca ou definida pelo programador.





Para se instanciar um objeto utiliza-se o método **Construtor**.

Construtores e Destrutores

- Construtor (mesmo nome da classe)
  - Todo o objeto deve ser instanciado (criado) através da ativação do método construtor.
- Destrutor (finalize)
  - O destrutor é o inverso do construtor, ele é ativado automaticamente quando o objeto está sendo destruído a fim de liberar a memória ocupada pelo mesmo.
- *Garbage Collection* (Coleta de Lixo)
  - O mecanismo de gerência automática de memória que destrói o objeto quando ele não está mais sendo usado.

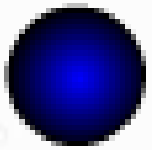
Classe Circulo - C++
Atributos

```
class Circulo {
public:
    int centroX, centroY;
    int raio;
};
```

```
int main()
{
    Circulo circ;
    circ.centroX = 5;
    circ.centroY = 3;
    circ.raio = 10;

    cout << "Circulo: centro( " << circ.centroX << ", "
         << circ.centroY << " ), raio " << circ.raio
         << endl;

    return 0;
}
```

Objeto `cout` e operador `<<`

- `cout`
  - objeto da biblioteca C++ que representa a saída padrão
- operador `<<`
  - operador de inserção
  - quando aplicado a uma stream, insere sequência na stream



Objeto em Java

- A instanciação do objeto se dá através do comando **`new`**.

- Quando o objeto é instanciado é acionado um método especial denominado construtor que tem o mesmo nome da classe.

Classe Circulo - Java
Atributos

```java
package pt.c02oo.s02classe.s01circulo01;

public class Circulo {
    int centroX, centroY;
    int raio;
}
```

```java
package pt.c02oo.s02classe.s01circulo01;

public class AppCirculo01 {
  public static void main(String args[]) {
      Circulo circ = new Circulo();
      circ.centroX = 5;
      circ.centroY = 3;
      circ.raio = 10;

      System.out.println("Circulo: centro(" + circ.centroX + ", " +
                         circ.centroY + "), raio " + circ.raio);
    }
}
```

Auto-referência e `this`

- Sem o this:

```
Circulo(int pCentroX, int pCentroY, int pRaio) {
    centroX = pCentroX;
    centroY = pCentroY;
    raio = pRaio;
}
```

- Com o this

```
Circulo(int centroX, int centroY, int raio) {
    this.centroX = centroX;
    this.centroY = centroY;
    this.raio = raio;
}
```

Auto-referência e `this`

- Para realizar referência a si próprio o objeto pode usar a referência **`this`**

```
public class Circulo {
    int centroX, centroY;
    int raio;

    Circulo(int pCentroX, int pCentroY, int pRaio) {
        centroX = pCentroX;
        centroY = pCentroY;
        raio = pRaio;
    }
```

Estudo de Caso

Bastião

Atributos

idade (1 a 3 anos)

estado (acordado, dormindo)

Métodos

aparecer, crescer, dormir e acordar

Estudo de Caso

Classe `Bastiao` no arquivo `Bastiao.java`

Classe `Principal` no arquivo `Principal.java`

Instanciação

Declaração da Referência

```
Bastiao theBastian;
```

Instanciação do Objeto
(chamada do construtor)

```
theBastian = new Bastiao();
```

Chamada de Método

```
theBastian.aparece();
```

Atributos Estáticos

- Recebem o prefixo static
- Atributos que pertencem à classe
  - mesmo valor do atributo compartilhado por todos os objetos

Métodos Estáticos

- Recebem o prefixo `static`
- Métodos de classe
  - não exigem instância para serem acionados
  - só podem acessar os atributos estáticos
  - só podem acionar outros métodos estáticos

Objetos da Biblioteca Java

Vetor

- Objeto especial do Java

- Declaração

```
<tipo>[] <declaração1>, ..., <declaraçãon>;
<tipo> <declaração1>[], ..., <declaraçãon>[];
```

  - <declaração>
    - Sintaxe: <nome> = <inicialização>
    - Chaves são usadas para inicializar cada dimensão
    - Ex.: `int primos[] = {1, 2, 3, 5, 7};`

Vetor

- A inicialização inline instancia automaticamente um objeto
- Quando a inicialização não é inline o vetor ou matriz precisa ser instanciado com new

```
<nome> = new <tipo>[<tamanho>]
```

  - Ex.:

```
int primos[];
primos = new int[5];
```

Estruturas de Dados Dinâmicas
Vector e ArrayList

- vetores dinâmicos
- Vector – sincronizado
  - mais seguro, mais lento
- ArrayList – não sincronizado
  - menos seguro, mais rápido
- Métodos
  - size()
  - add(<elemento>)
  - get(<posição>) → <elemento>

Estruturas de Dados Dinâmicas
Hashtable

- Tabela hash
- Métodos:
  - put(<chave>, <valor>)
  - get(<chave>) → <valor>

Estruturas de Dados Dinâmicas
Stack

- Pilha
- Métodos:
  - push(<elemento>)
  - pop() → <elemento>

JavaScript e Protótipos

JavaScript

Objeto com Método (1)

```
function exemploObjetoMetodo1() {
  var circulo = {
     centroX : 5,
     centroY : 3,
     raio : 10,
     area : function() {
        return 3.1416 * this.raio * this.raio;
     }
  };

  alert("Circulo: centro(" + circulo.centroX + ", " +
        circulo.centroY + "), raio " + circulo.raio + ",
        area " + circulo.area());
}
```

JavaScript

Objeto com Método (2)

```
function exemploObjetoMetodo3() {
  var circulo = {};

  circulo.centroX = 5;
  circulo.centroY = 3;
  circulo.raio = 10;
  circulo.area = function() {
            return 3.1416 * this.raio * this.raio;
              }

  alert("Circulo: centro(" + circulo.centroX + ", " +
        circulo.centroY + "), raio " + circulo.raio +
        ", area " + circulo.area());
}
```

JavaScript

Objeto com Método (2)

```
function exemploObjetoMetodo3() {
  var circulo = {};

  circulo.centroX = 5;
  circulo.centroY = 3;
  circulo.raio = 10;
  circulo.area = function() {
            return 3.1416 * this.raio * this.raio;
              }

  alert("Circulo: centro(" + circulo.centroX + ", " +
        circulo.centroY + "), raio " + circulo.raio +
        ", area " + circulo.area());
}
```

JavaScript
Protótipo (1)

```
function Circulo01() {}

Circulo01.prototype.centroX = 5;
Circulo01.prototype.centroY = 3;
Circulo01.prototype.raio = 10;

Circulo01.prototype.area = function() {
                             return 3.1416 * this.raio * this.raio;
                           };

function exemploPrototipo01() {
  var circulo = new Circulo01();

  console.log("Circulo: centro(" + circulo.centroX + ", " +
              circulo.centroY + "), raio " + circulo.raio +
              ", area " + circulo.area());

  console.log(Circulo01.prototype);
}
```

JavaScript
Protótipo (2)

```
function Circulo02() {}

Circulo02.prototype = {
  centroX: 5,
  centroY: 3,
  raio: 10,
  area: function() {
          return 3.1416 * this.raio * this.raio;
        }
};

function exemploPrototipo02() {
  var circulo = new Circulo02();

  console.log("Circulo: centro(" + circulo.centroX + ", " +
              circulo.centroY + "), raio " + circulo.raio +
              ", area " + circulo.area());

  console.log(Circulo01.prototype);
}
```

JavaScript
Protótipo (3)

```
function Circulo03() {
   this.centroX = 5;
   this.centroY = 3;
   this.raio = 10;

   this.area = function() {
                 return 3.1416 * this.raio * this.raio;
               }
}

function exemploPrototipo03() {
  var circulo = new Circulo03();

  console.log("Circulo: centro(" + circulo.centroX + ", " +
              circulo.centroY + "), raio " + circulo.raio +
              ", area " + circulo.area());

  console.log(Circulo02.prototype);
}
```

Referências Bibliográficas

- Almeida, Charles Ornelas , Guerra, Israel; Ziviani, Nivio (2010) **Projeto de Algoritmos** (transparências aula).
- Bloom, Paul (2007) **Introduction to Psychology** – transcrição das aulas (aula 17). Yale University.
- Ferreira, Aurélio B. H. (1989) **Minidicionário da Língua Portuguesa**. Rio de Janeiro, Editora Nova Fronteira.
- Houaiss, Instituto Antônio. **Dicionário Houaiss da língua portuguesa** (2006) Editora Objetiva, Março.
- IBM - International Business Machines Corporation. **IBM Smalltalk Tutorial** [Online] http://www.wi2.uni-erlangen.de/sw/smalltalk/
- Liskov, Barbara; Zilles, Stephen. **Programming with abstract data types** (1974) ACM SIGPLAN Notices, 9 (4) p. 50.

Referências Bibliográficas

- Meyer, Bertrand (1997) **Object-Oriented Software Construction - Second Edition**. USA, Prentice-Hall, Inc.
- Miller, Robert (2004) **6.831 User Interface Design and Implementation (lecture notes)**. MIT OpenCourseware.
- Rocha, Heloisa Vieira da, Baranauskas, Maria Cecilia Calani (2003) **Design e Avaliação de Interfaces Humano-Computador.** NIED/UNICAMP.
- Santos, L. R., & Hood, B. M. (2009). **Object representation as a central issue in cognitive science**. The Origins of Object Knowledge: The Yale Symposium on the Origins of Object & Number Representation. Oxford: Oxford University Press.
- Shaw, M. **Abstraction Techniques in Modern Programming Languages** (1984) IEEE Software, 1, 4, 10-26.

Referências Bibliográficas

- Tenenbaum, Aaron M.; Langsam, Yedidyah; Augenstein, Moshe J. **Data Structures Using C** (1990) Prentice Hall, Upper Saddle River, NJ.

André Santanchè

http://www.ic.unicamp.br/~santanche

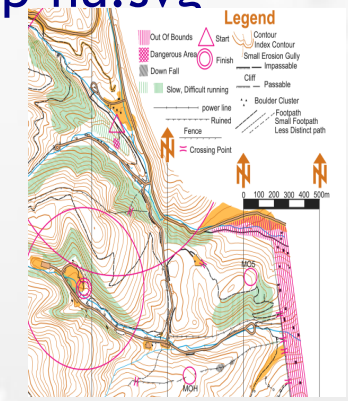
License

- These slides are shared under a Creative Commons License. Under the following conditions: Attribution, Noncommercial and Share Alike.
- See further details about this Creative Commons license at: http://creativecommons.org/licenses/by-nc-sa/3.0/

Imagens Externas

- Havang(nl) [http://commons.wikimedia.org/wiki/User:Havang%2
  url (ver licença específica):
  http://commons.wikimedia.org/wiki/File:Bomenpark_Meijhorst,_Nijmegen_%28

- Eric Gaba [http://commons.wikimedia.org/wiki/User:Sting]
  url (ver licença específica):
  http://commons.wikimedia.org/wiki/File:Easter_Island_map-hu.svg



- Kharker [http://en.wikipedia.org/wiki/User:Kharker]
  url (ver licença específica):
  http://commons.wikimedia.org/wiki/File:Ardf_map.png