

Programação Orientada a Objetos

Orientação a Objetos, Objetos e Classes

André Santanchè
Institute of Computing - UNICAMP
Março 2011

Orientação a Objetos

Um pouco de história

- **SIMULA 67**
 - Primeira Linguagem Orientada a Objetos
- **Smalltalk**
 - Projeto Dynabook
 - “Este ‘Dynabook’ foi baseado na visão de computadores pessoais baratos do tamanho de um caderno, tanto para adultos quanto crianças, com a capacidade de lidar com todas as suas respectivas necessidades de informação”. [KRE98]

Problema, Abstração e Tipo Abstrato de Dados

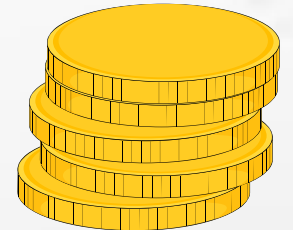
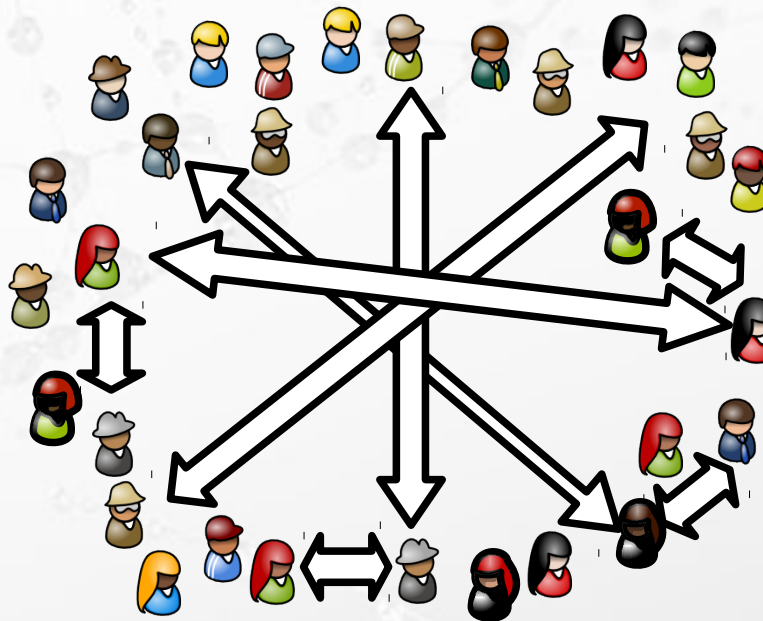
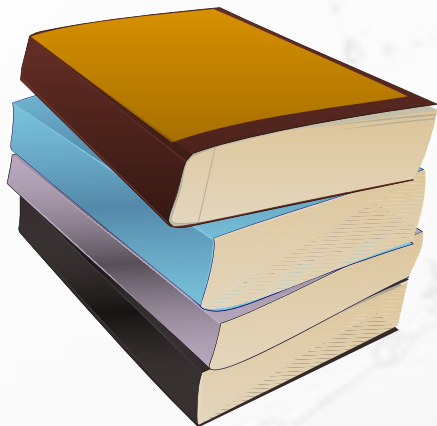
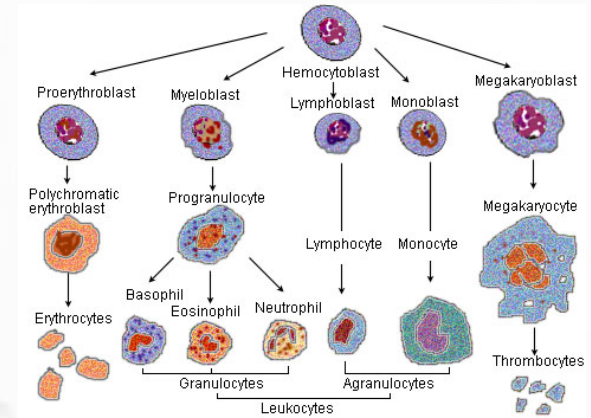
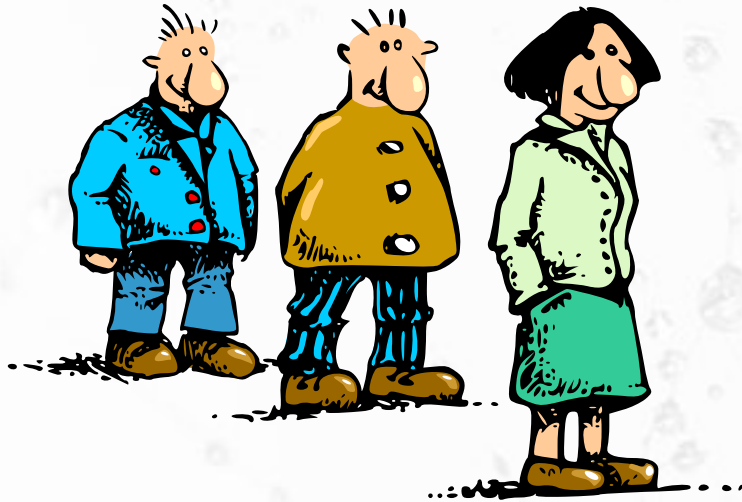
Problema x Abstração

- “Para resolver um problema é necessário escolher uma abstração da realidade”
(Almeida, 2010)

Abstração

- **“processo mental que consiste em escolher ou isolar um aspecto determinado de um estado de coisas relativamente complexo, a fim de simplificar a sua avaliação, classificação ou para permitir a comunicação do mesmo” (Houaiss, 2006)**
- **Abstrações ajudam a gerenciar a complexidade do software (Shaw, 1984)**

Abstrações do Dia a Dia



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)

Objetos, Classes e como vemos o mundo

Objetos

- Montanha

Objetos

- Estação rodoviária

Noção de Objeto

- Psicologia do desenvolvimento:
 - Quando crianças representam objetos como entidades permanentes?
 - Que persistem:
 - Através do tempo e espaço
 - À oclusão

(Santos & Hood, 2009)

Noção de Objetos

- **Objetos permanecem?**
 - “Of course, the concept of object permanence itself is really a misnomer, as all objects comprise energy in continuous states of change.” (Santos & Hood, 2009)
- **Estereótipos**
 - “[...] nervous systems were developed via natural selection to represent objects so that organisms may interact with the external world in an adaptive way, and thus, brains are built to capture what is functionally relevant about objects.” (Santos & Hood, 2009)

Noção de Objetos

- **Objetos necessários**

- “One of the most functionally relevant aspects of physical objects is the fact that they persist—standardly speaking, objects do not go in and out of existence and, thus, it is important that an organism be able to represent their continued presence even when they cannot be directly perceived or apprehended.” (Santos & Hood, 2009)

Noção de Objetos

- Existência independente do observador
 - “[...] nervous systems were developed via natural selection to represent objects so that organisms may interact with the external world in an adaptive way, and thus, brains are built to capture what is functionally relevant about objects.”
(Santos & Hood, 2009)

Objetos e Memória

Memória de Curta Duração (Trabalho)

- Armazena:
 - produtos intermediários do pensamento
 - representações produzidas pelo Sistema Perceptual
- Operações mentais:
 - obtém operandos
 - deixam resultados intermediários

(Rocha, 2003)

Chunks

- “Conceitualmente a MCD é constituída de chunks: elementos ativados da MLD, que podem ser organizados em unidades maiores.”
(Rocha, 2003, p. 55)

Estereótipos

- Estereótipo
 - “We tend to use the term to refer to information we have about categories and intuitions we have about the typicality, our frequency of certain features of categories.” (Bloom, 2007)

Estereótipos

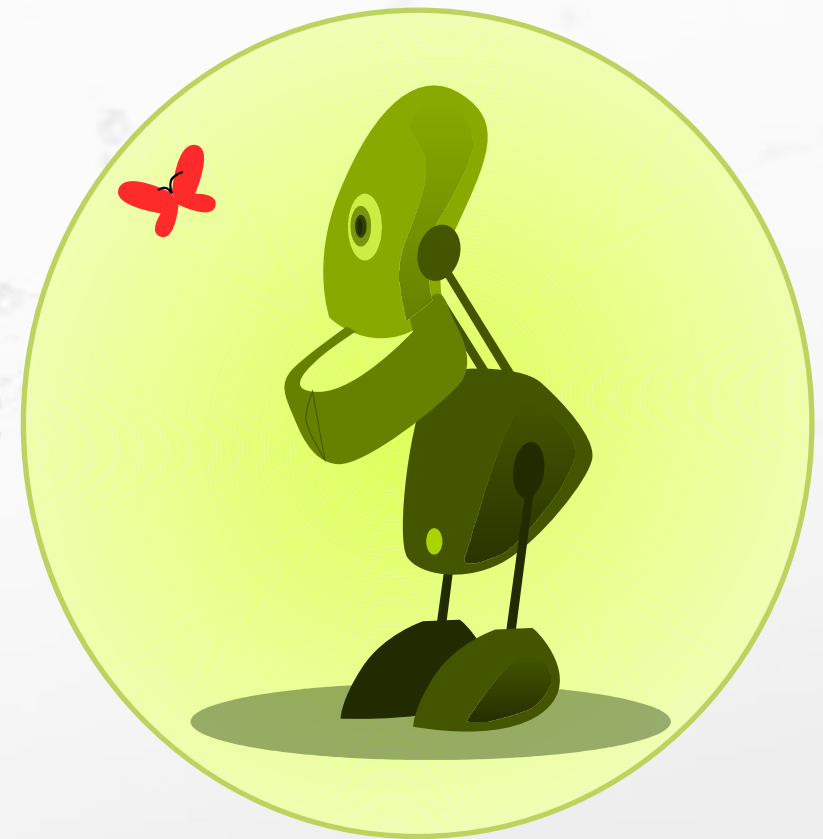
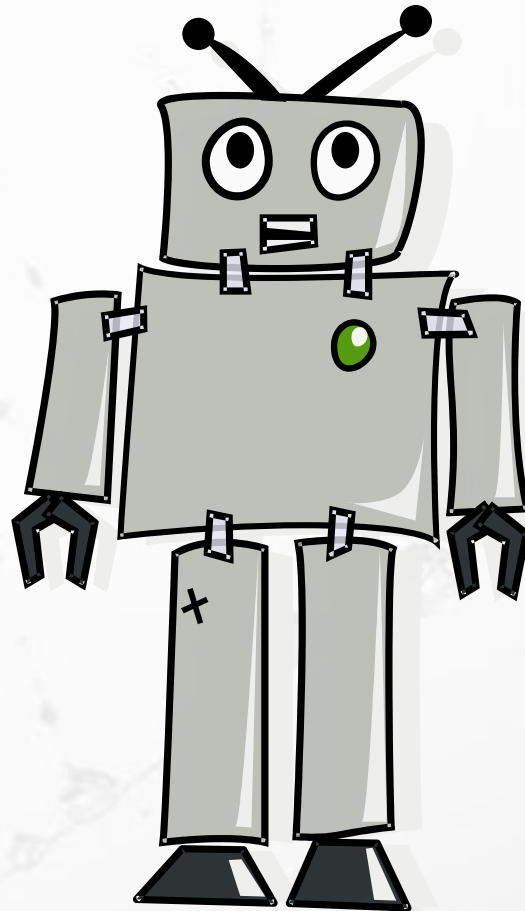
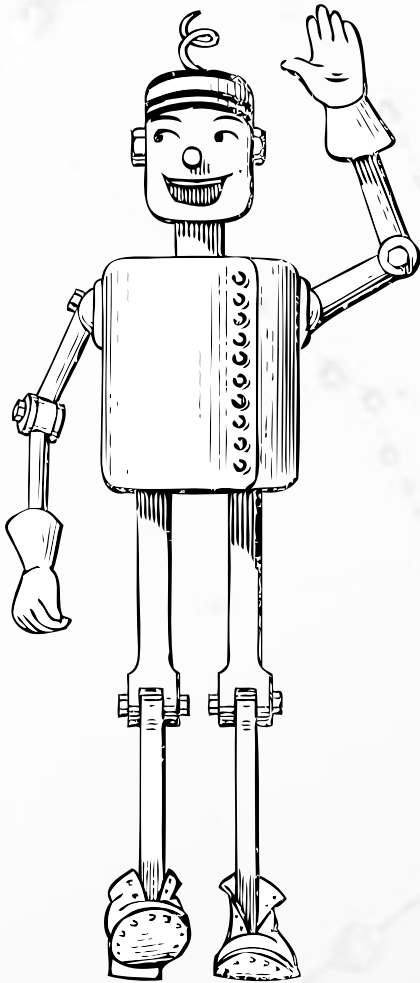
- Essencial para sobrevivência
 - And it turns out that collecting information about categories is essential to our survival. We see novel things all the time and if we were not capable of learning and making guesses, educated guesses, about these novel things we would not be able to survive. So, when you see this object over here you categorize it as a chair and you recognize that you could probably sit on it.” (Bloom, 2007)

Estereótipos

- Generalização

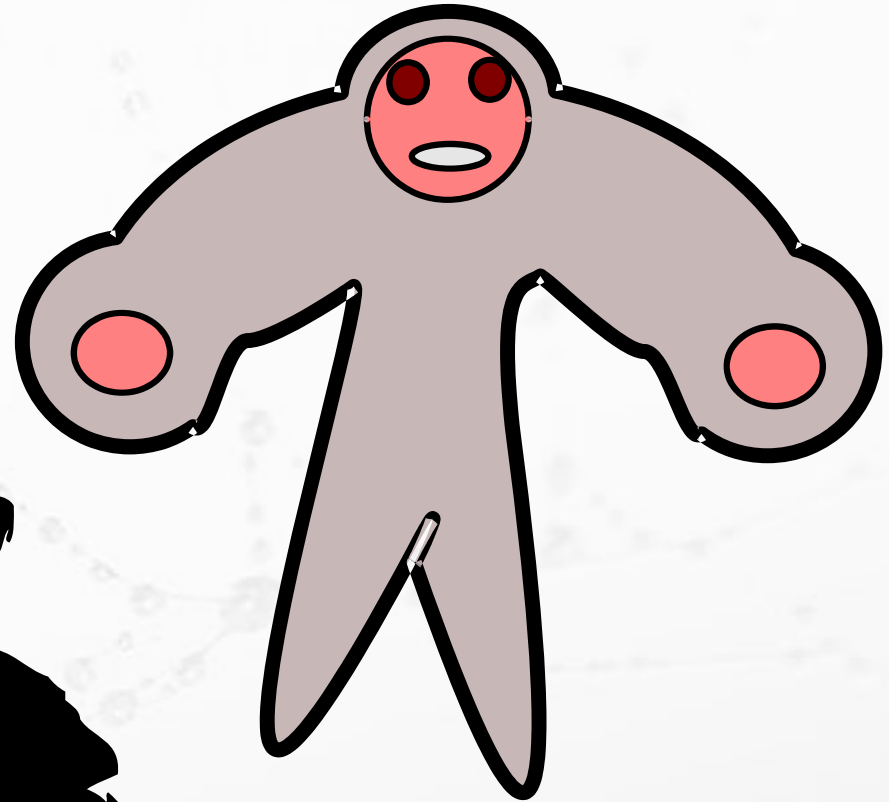
- “And if you were suddenly stripped of your ability to make generalizations, you'd be at a loss. You wouldn't know what to eat, how to interact. So, some sort of ability to record information and make generalizations is absolutely essential to making it through life.” (Bloom, 2007)

Desafios da Representação Compartilhada Estereótipos



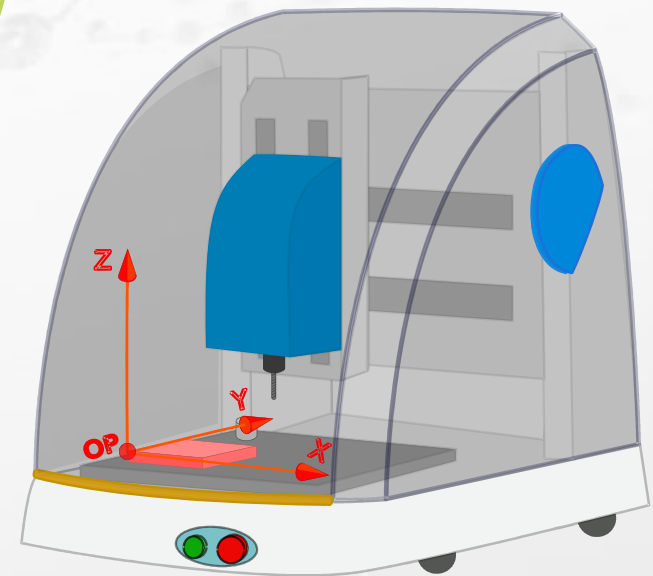
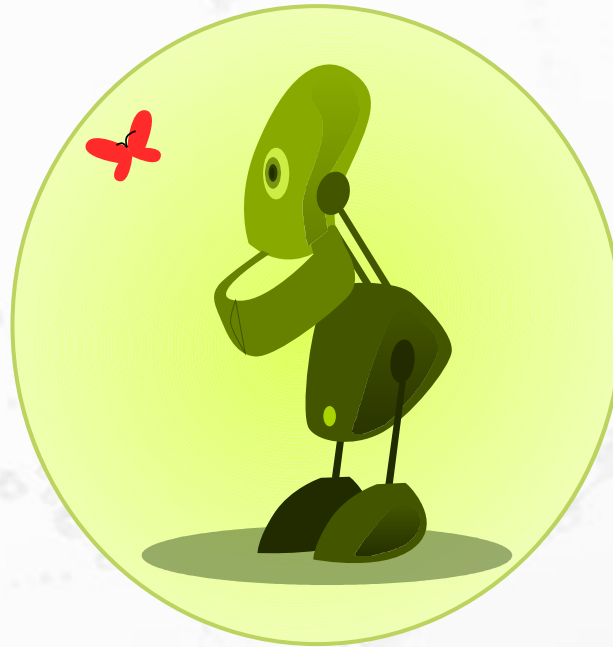
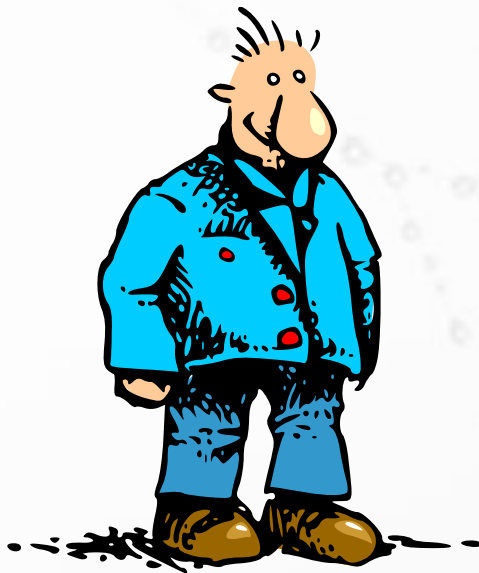
Desafios da Representação Compartilhada

Estereótipos



Estereótipos Abstrações Humanas

- São o mundo real ou descrevem o mundo real?



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

```
#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 é 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

```
int combinacoes;  
  
void inicializa(int *combinacoes) {  
    *combinacoes = 1;  
}  
  
int proximoNumeroCombinacoes(int *combinacoes) {  
    *combinacoes *= 2;  
    return *combinacoes;  
}  
  
int main()  
{  
    int bits;  
    inicializa(&combinacoes);  
    for (bits = 1; bits <= 8; bits++)  
        printf("%d bits = %d combinacoes", 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

```
#include "bits05module.h"

static int combinacoes;

void inicializa()
{
    combinacoes = 1;
}

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

Modularização Sucessiva

Quinta Versão - bits05.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 é 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 Versão

Java

O módulo é transformada em uma classe

Modularização Sucessiva

Sexta Versão - Classe

```
package bits;

public class Bits06Class
{
    private int combinacoes;

    public void inicializa()
    {
        combinacoes = 2;
    }

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


● Objeto

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

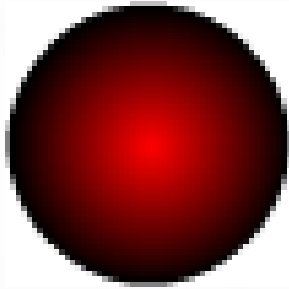
- Objetos são caracterizados por:
 - identidade;
 - atributos;
 - comportamento.

Exemplo de Objeto

Esfera Vermelha

Objeto Esfera

Atributos (nome, valor)



(**peso**, 200 g)

(**raio**, 60 cm)

(**elasticidade**, alta)


(**cor**, vermelha)

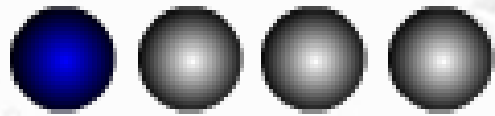
Comportamento

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	



Classe

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

- Classificação de Carl Linné



Amphibia



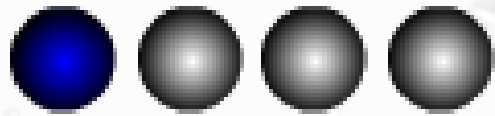
Reptilia



Aves

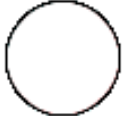





Mammalia



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.

Classe	Objeto	Objeto	Objeto
 peso raio cor	 peso: 200 g raio: 60 cm cor: vermelha	 peso: 200 g raio: 60 cm cor: azul	 peso: 50 g raio: 30 cm cor: verde

Objetos e Classes

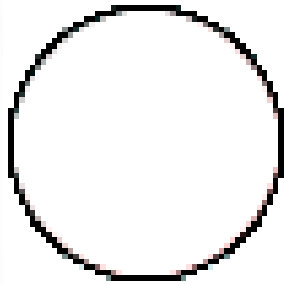
- Os objetos são organizados/divididos em grupos chamados classes.
- Objetos da mesma classe têm:
 - o mesmo conjunto de atributos (os valores dos atributos podem ser diferentes);
 - o mesmo conjunto de métodos.

Exemplo de Classe

Esfera

Classe Esfera

Atributos (nome, tipo)



(**peso**, real)

(**raio**, real)

(**elasticidade**, string)

(**cor**, color)

Comportamento

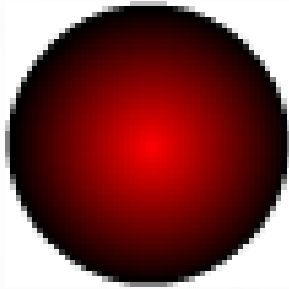
aumentar, diminuir, se mover

Exemplo de Objeto

Esfera Vermelha

Objeto Esfera

Atributos (nome, valor)



(**peso**, 200 g)

(**raio**, 60 cm)

(**elasticidade**, alta)

(**cor**, vermelha)

Comportamento

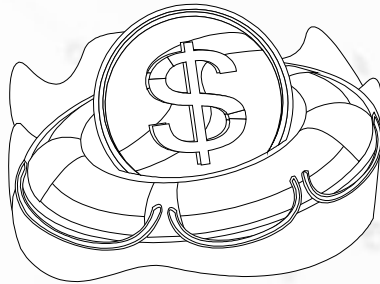
aumentar, diminuir, se mover

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)



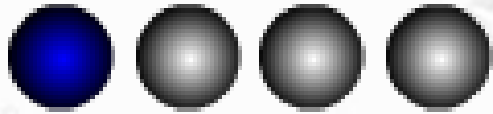
(valor, R\$ 150)

(número de parcelas, 3)

(percentual de juros, 1%)

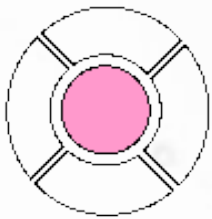
Comportamento

calcula parcela

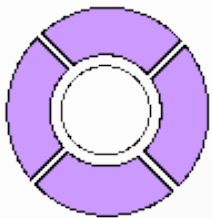


Classe

- Em Programação Orientada ao Objeto:



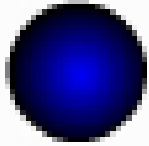
Atributos: dados que pertencem a cada instância da classe (objeto); são definidos sob a forma de variáveis.



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

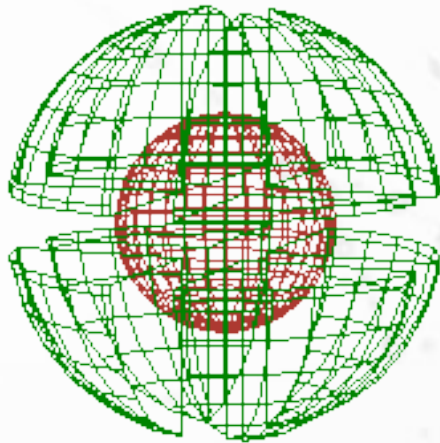
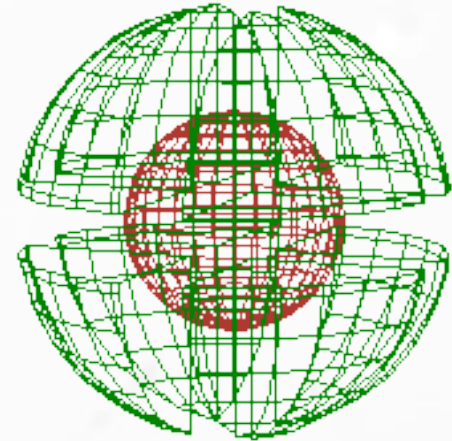
Objeto instância de Classe

- 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.

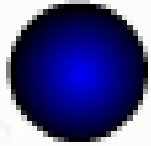


Para se instanciar um objeto utiliza-se o método **Construtor**.



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.



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.

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

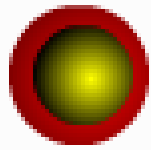
```
Bastiao theBastian;
```

Instanciação do Objeto (chamada do construtor)

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

Chamada de Método

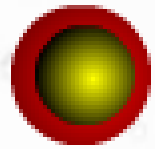
```
theBastian.aparece ();
```



Princípios do Paradigma Encapsulamento

- "Objetos do mundo real encapsulam em si os próprios atributos, quer sejam descritivos, partes componentes ou funções." (Meyer, 1997)
- **Objetos e mensagens**
 - “Na programação orientada a objetos, os objetos comunicam-se entre si através de mensagens. A única coisa que um objeto conhece sobre outro objeto é a sua interface de comunicação. Os dados e a lógica de cada objeto são mantidos escondidos dos outros objetos. Em outras palavras, a interface encapsula o código e os dados do objeto”. (IBM)

Encapsulamento



Interface x Implementação

- **Interface:** descreve como as partes do objeto se relacionam com o exterior.
- **Implementação:** dados e código que implementam o comportamento dos objetos da classe; esta parte não é visível externamente.

Encapsulamento

Níveis de Acesso

- **Privada:** não visível a nenhuma classe externa; visível apenas dentro da classe.
- **Pública:** completamente visível internamente e para outras classes e elementos externos.
- **Protegida:** não visível a classes e elementos externos; visível apenas dentro da classe e para seus herdeiros.

Encapsulamento

Atributos Privados e Métodos Públicos

- Na POO uma classe possui, em geral, os **atributos privados** e os **métodos podem ser públicos**, tornando o objeto como uma caixa preta onde só aparece o suficiente para que o programa possa utilizá-lo.

Java

Níveis de Acesso

- **Privada (private):** visível exclusivamente dentro da classe.
- **Pública (public):** completamente visível dentro e fora da classe.
- **Protegida (protected):** visível apenas dentro da classe, pelos seus herdeiros e pelas classes no mesmo pacote.

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, Yedidiah; 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/>