

An abstract, light blue geometric pattern consisting of interconnected lines and star-like shapes, resembling a complex network or a stylized molecular structure, is positioned in the upper left corner of the slide.

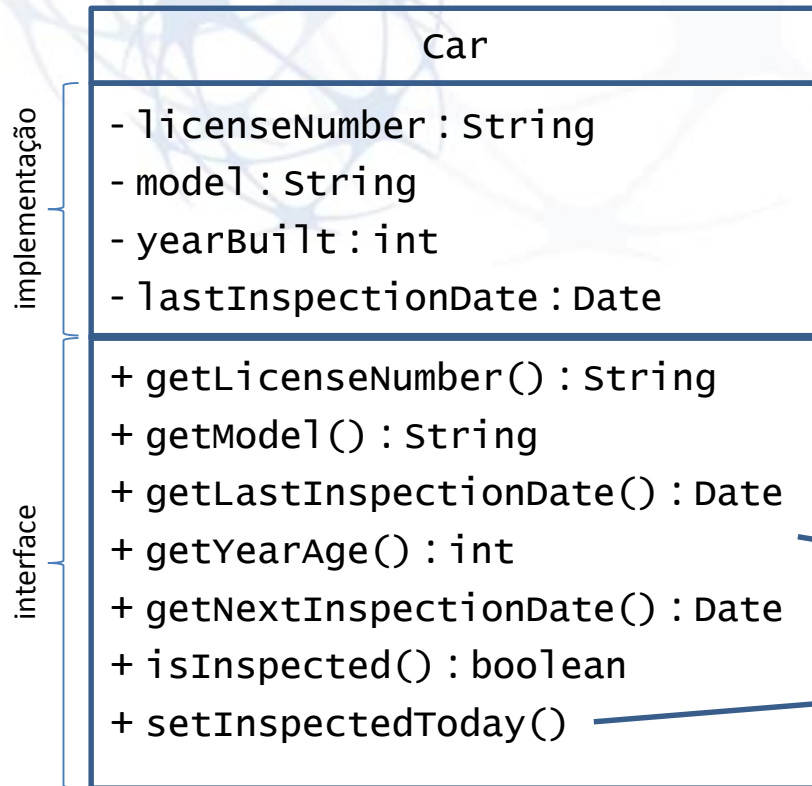
Classes e Objetos

Encapsulamento

Classes

- Uma classe é identificada por um nome
 - O nome deve reflectir o que os objectos da classe representam (no singular)
 - p.e. Point, Set, Person, Game, Board, Player
 - **Convenção (Java)**: deve começar com com letra maiúscula
- Uma classe (que não seja uma **classe pacote**) é composta essencialmente por **atributos**, **constructores**, e **métodos de instância**

Classes em Java



- Definem conjunto de características (propriedades e operações) comuns a todas as suas instâncias.

Classes e objetos

- Classe
 - Modelo para a criação de objetos que partilham um conjunto de características comuns
 - Atributos ↔ Variáveis (Java)
 - Operações ↔ Métodos (Java)
- Objeto
 - Instância de uma classe
 - É criado e manipulado durante a execução do programa
 - Tem identidade e estado próprios

Objetos em Java

johnsCar : Car

licenseNumber = 00-aa-00

model = VW-GTI-TDI-SLK

yearBuilt = 2005

lastInspectionDate = 2009-11-20

- Instâncias de uma classe com valores específicos nos seus atributos e, por isso, com propriedades bem definidas.

Membros de uma classe

- Atributos
 - Variáveis que definem o estado dos objectos da classe
- Construtor(es)
 - Métodos particulares que tem por objectivo inicializar os atributos de um novo objeto
- Métodos de instância
 - Métodos que executam acções nos objectos da classe

Atributos

- Atributos são variáveis cujos valores caracterizam um objecto
 - Representam o estado do objecto
 - Cada objecto detém as suas variáveis
- Exemplos:
 - x e y como atributos de objectos *Ponto*
 - *cardinalidade* como atributo de objectos *Conjunto*
 - *nome* como atributo de objectos *Pessoa*

Objetos (instâncias de classes)

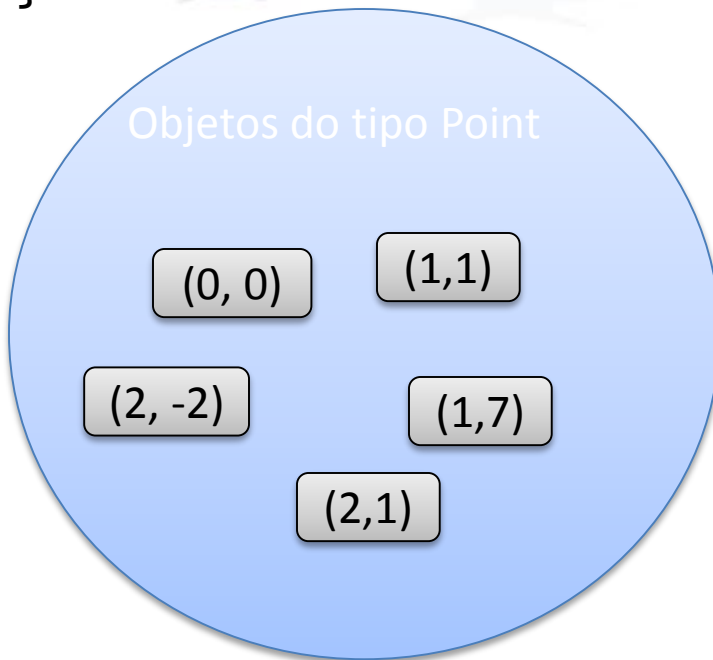
```
public class Point {  
    int x;  
    int y;  
    ...  
}
```

atributos

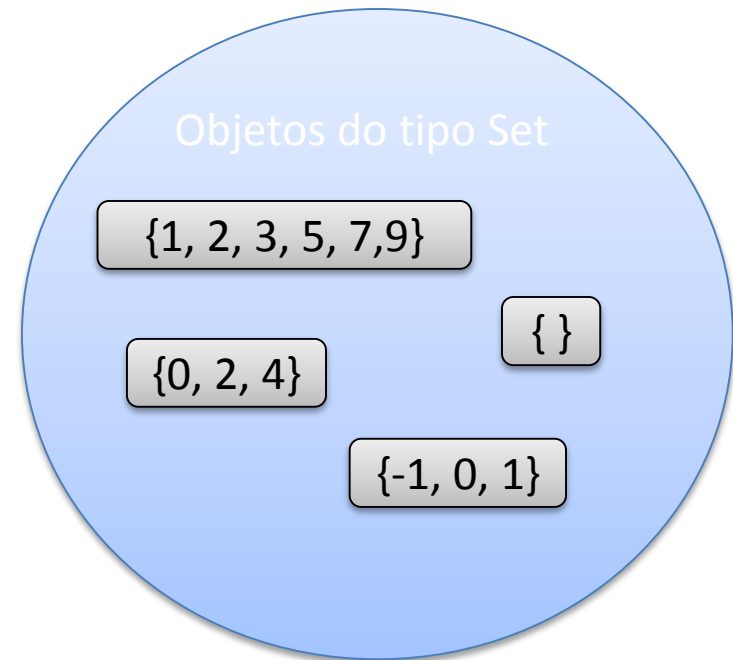
```
public class Set {  
    int[] elements;  
    ...  
}
```

atributos

Objetos do tipo Point



Objetos do tipo Set



Métodos construtores

- Um método **construtor** de uma classe é um método particular cujo propósito é inicializar os atributos de objectos dessa classe em criação
 - Pode haver vários construtores, porém com parâmetros diferentes
 - O construtor não tem tipo de devolução
 - Colocam objectos num estado inicial válido
 - O construtor chama-se usando **new** }

É boa prática definir atributos como sendo privados (private).
A ver depois...

```
public class Point {  
    private int x;  
    private int y;
```

```
    public Point  
        x = 0;  
        y = 0;
```

```
    public Point int x, int y  
        x  
        y
```

this : utilizado para desambiguar
no caso das variáveis locais terem
o mesmo nome que os atributos

Criação de objetos

- Operador **new** cria novos objetos

new ClasseDoNovoObjecto(argumentos)

- Os argumentos têm de ser compatíveis com um dos construtores

– Exemplos:

`new Point()`

`new Point(1, -2)`

```
public Point() {  
    x = 0;  
    y = 0;  
}
```

```
      1    -2  
      ↓    ↓  
public Point(int x, int y) {  
    this.x = x;  
    this.y = y;  
}
```

Inicializações automáticas

- Atributos e elementos de um vector que sejam de um tipo primitivo são inicializados com o **valor por omissão**:
 - int - 0
 - double - 0.0
 - boolean - false
 - ...
- Atributos e elementos de um vector que sejam de um tipo de referência são inicializados a **null**

Métodos de instância

- Implementam operações a realizar sobre um objeto que definem o seu comportamento
- Podem ser **funções** (calculam e devolvem um resultado) ou **procedimentos** (realizam uma *ação*)
- Podem ter parâmetros
- Podem alterar (**modificadores**) ou não (**inspetores**) o estado do objeto
- Em geral as funções são inspetores e os modificadores são procedimentos

Métodos inspetores

- Métodos inspetores são funções que permitem obter o valor de atributos
- Não tem que haver uma correspondência de um para um entre atributos e inspetores
- A decisão de incluir um inspetor para determinado atributo depende do contexto em questão

```
public class IdCard {  
    private String firstName;  
    private String lastName;  
    private int id;  
  
    ...  
  
    public String fullName() {  
        return firstName + " " + lastName;  
    }  
  
    public int id() {  
        return id;  
    }  
  
    ...  
}
```

Também poderia usar-se
`getId()`

Métodos modificadores

- Caso seja apropriado, uma classe pode incluir métodos para modificar o valor dos atributos

```
public class IdCard {  
    private String firstName;  
    private String lastName;  
    private int id;  
    ...  
  
    public void setFirstName(String name) {  
        firstName = name;  
    }  
  
    public void setLastName(String name) {  
        lastName = name;  
    }  
    ...  
}
```

No BI, o número nunca muda...

Métodos: funções e procedimentos

- Funções

- Conjunto de instruções, com interface bem definida, que efectua um dado cálculo
- Devolvem explicitamente um resultado ao exterior
- Não devem efectuar qualquer alteração ao estado do objeto

- Procedimentos

- Conjunto de instruções, com interface bem definida, que realiza uma determinada acção (normalmente, alteram o estado do objecto)
- Não devolvem explicitamente um resultado ao exterior

Função

```
public class Nome {  
    private tipo atributo;  
    ...  
}
```

atributos não devem ser modificados pela função

assinatura { public *tipo* nome(*parâmetros*) {
 instruções
 return *expressão*;
}

corpo da função

Procedimento

```
public class Nome {  
    private tipo atributo;  
    ...  
}
```

*atributos podem ser
modificados pelo procedimento*

```
assinatura { public void nome(parâmetros) {  
    instruções  
}  
}
```

corpo do procedimento

Operações e métodos em Java

- Operações
 - Parte da interface da classe
 - Invocam-se
- Métodos
 - Parte da implementação da classe
 - Executados quando se invoca a operação correspondente
- Uma única operação pode ser implementada por vários métodos

Como? Usando polimorfismo de subtipos, que se verá mais tarde.

Operações em Java: boas práticas

- Cada operação deve ter um objetivo (uma função) único e bem definido
 - Operações inspetoras – Nome reflete aquilo que devolvem
 - Outras operações – Nome reflete a ação que realizam
- Uma operação não deve tentar ser simultaneamente inspetora e modificadora (função e procedimento)

Exemplo: calculadora

```
public class Calculator {  
    atributo { private int value;  
    construtor { public Calculator() {  
        value = 0;  
    }  
    função { public int value() {  
        return value;  
    }  
    procedimentos { public void set(int newValue) {  
        value = newValue;  
    }  
    public void clear() {  
        value = 0;  
    }  
    public void add(int term) {  
        value = value + term;  
    }  
    ...  
}
```

```
calculator c = new Calculator();
```

c

```
c.value();
```

```
c.set(5);
```

```
c.clear();
```

```
c.add(10);
```



“Objeto calculadora”
(metáfora)

Encapsulamento

- “esconder” atributos e métodos de um objeto do exterior (clientes da classe)
- clara separação entre *interface* e *implementação*, permite:
 - Maior flexibilidade na evolução da implementação de uma classe
 - Maior controlo sobre a correta utilização dos objetos de uma classe

Encapsulamento

- Modificadores de acesso (para atributos e métodos):
 - **public** – permite acesso direto do exterior
 - **private** – não permite acesso direto do exterior, sendo possível apenas o acesso interno (no contexto dos métodos da classe)
 - (existem outros, a ver mais tarde)

Encapsulamento de atributos

- É considerado boa prática encapsular os atributos de um objeto

```
public class Rational {  
    private int numerator;  
    private int denominator;  
  
    ...  
}
```

```
Rational r = new Rational(1, 4);
```

```
r.denominator = 0;
```



*O compilador identifica um erro
ao aceder ao atributo externamente*

Encapsulamento

- Ponto num espaço bidimensional (exemplo)
 - Um conceito, duas formas de representar
 - Coordenadas cartesianas
 - Coordenadas polares
 - Classe para representar pontos
 - Uma interface, duas (ou mais) formas de implementar

```
public class Point {  
    private double abscissa;  
    private double ordinate;  
    ...  
}
```

```
public class Point {  
    private double radius;  
    private double angle;  
    ...  
}
```


Separação entre interface e implementação (encapsulamento)

```
public class Point {  
    private double abscissa;  
    private double ordinate;
```

```
    public Point(double abscissa, double ordinate) {  
        this.abscissa = abscissa;  
        this.ordinate = ordinate;  
    }
```

```
    public double abscissa() {  
        return abscissa;  
    }
```

```
    public double ordinate() {  
        return ordinate;  
    }
```

```
    public double radius() {  
        return Math.sqrt(abscissa*abscissa + ordinate*ordinate);  
    }
```

```
    public double angle() {  
        return Math.atan2(ordinate, abscissa);  
    }
```

```
}
```


```
Point p = new Point(1.2, 2.7);  
double abs = p.abscissa();  
double ord = p.ordinate();  
double rad = p.radius();  
double ang = p.radius();
```



Separação entre interface e implementação (encapsulamento)

```
public class Point {  
    private double radius;  
    private double angle;  
  
    public Point(double abscissa, double ordinate) {  
        radius = Math.sqrt(abscissa * abscissa + ordinate * ordinate);  
        angle = Math.atan2(ordinate, abscissa);  
    }  
  
    public double abscissa() {  
        return Math.cos(angle) * radius;  
    }  
  
    public double ordinate() {  
        return Math.sin(angle) * radius;  
    }  
  
    public double radius() {  
        return radius;  
    }  
  
    public double angle() {  
        return angle;  
    }  
}
```

```
Point p = new Point(1.2, 2.7);  
double abs = p.abscissa();  
double ord = p.ordinate();  
double rad = p.radius();  
double ang = p.radius();
```



Controlo sobre a utilização de um objeto (encapsulamento)

```
public class ContactList {
    int nextInsert;
    Contact[] contacts;

    public ContactList(int capacity) {
        nextInsert = 0;
        contacts = new Contact[capacity];
    }

    public boolean isFull() {
        return nextInsert == contacts.length;
    }

    public void insert(Contact c) {
        if(!isFull()) {
            contacts[nextInsert] = c;
            nextInsert++;
        }
    }
    ...
}
```

```
ContactList list = new ContactList(4);
list.insert(new Contact(..));
list.nextInsert = 5;
list.insert(new Contact(..));
```




O programa termina abruptamente com um erro! (ArrayIndexOutOfBoundsException)

Controlo sobre a utilização de um objeto (encapsulamento)

```
public class ContactList {  
    private int nextInsert;  
    private Contact[] contacts;  
  
    public ContactList(int capacity) {  
        nextInsert = 0;  
        contacts = new Contact[capacity];  
    }  
  
    public boolean isFull() {  
        return nextInsert == contacts.length;  
    }  
  
    public void insert(Contact c) {  
        if(!isFull()) {  
            contacts[nextInsert] = c;  
            nextInsert++;  
        }  
    }  
    ...  
}
```

```
ContactList list = new ContactList(4);  
list.insert(new Contact(..));  
list.nextInsert = 5;  
list.insert(new Contact(..));
```



O compilador identifica um erro ao aceder ao atributo externamente

Referências

- Y. Daniel Liang, "Introduction to Java Programming" 7th Ed. Prentice-Hall, 2010.

Sumário

- Classes e Objetos
- Encapsulamento