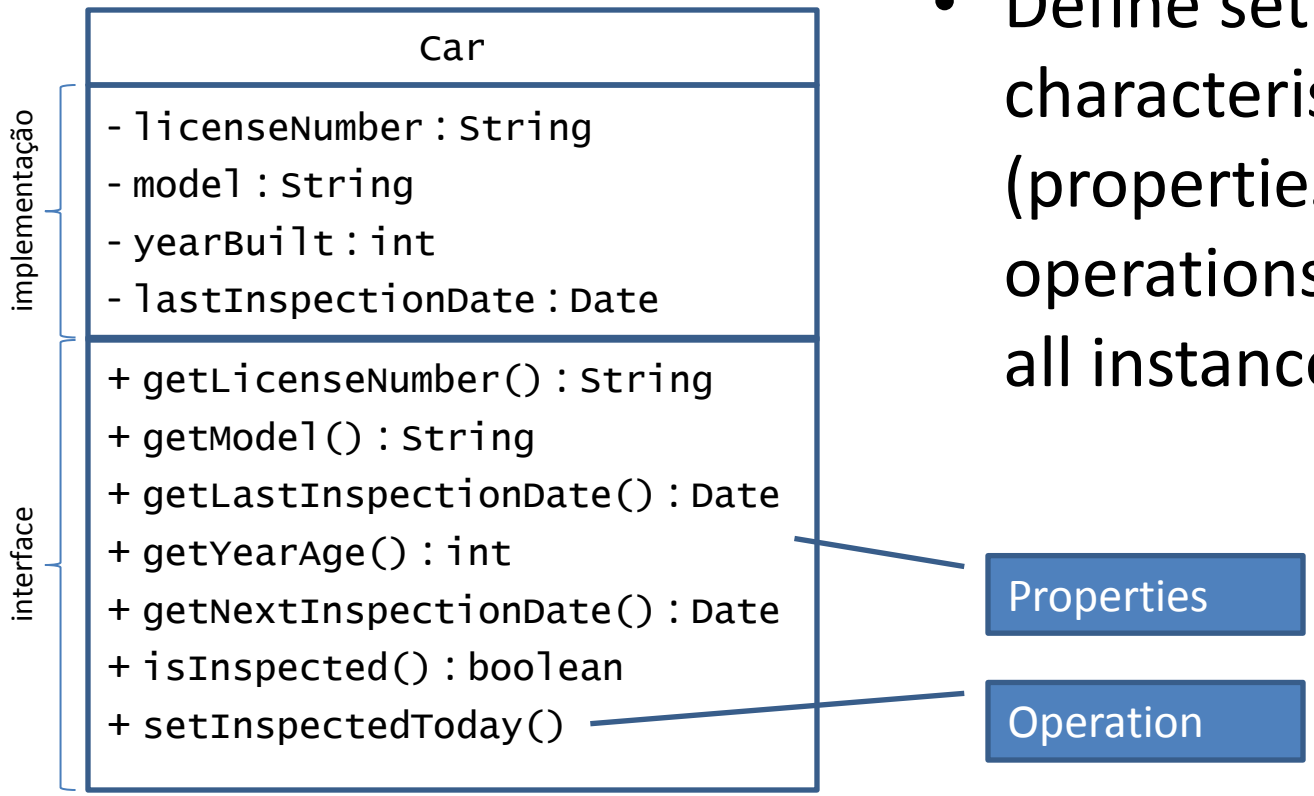
Classes and Objects

Classes

- A class is identified by a name
  - The name should reflect what the objects stand for: Point, Set, Person, Game, Board, Player
    - Convention (Java): Initial capitalized

- A class (Except for package classes) has attributes, constructors, and (instance) methods

Classes in Java

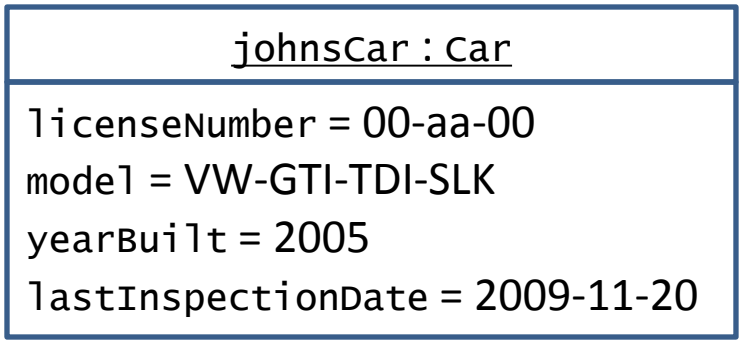


- Define set of characteristics (properties and operations) common to all instances

Class and object

- Class
  - Model for the creation of objects that share common characteristics
    - Attributes ↔ Variables (Java)
    - Operations ↔ Methods (Java)
- Object
  - Instance of a class
  - Created and manipulated during execution
  - Has identity and state

Objects in Java



- Instances of a class with specific values in their attributes

Members of a class

- Attributes
  - Variables that define each object's state
- Constructor(s)
  - Methods that initialize objects
- Instance methods
  - Methods that execute actions on the objects

Attributes

- Attributes are variables whose values define the object's state
  - Attribute values define the object
  - Each object has its own set of attributes

- Examples:
  - *x* and *y* are attributes of the class *Point*
  - *numberOfElements* is an attribute of the *Set*
  - *name* is an attribute of *People*

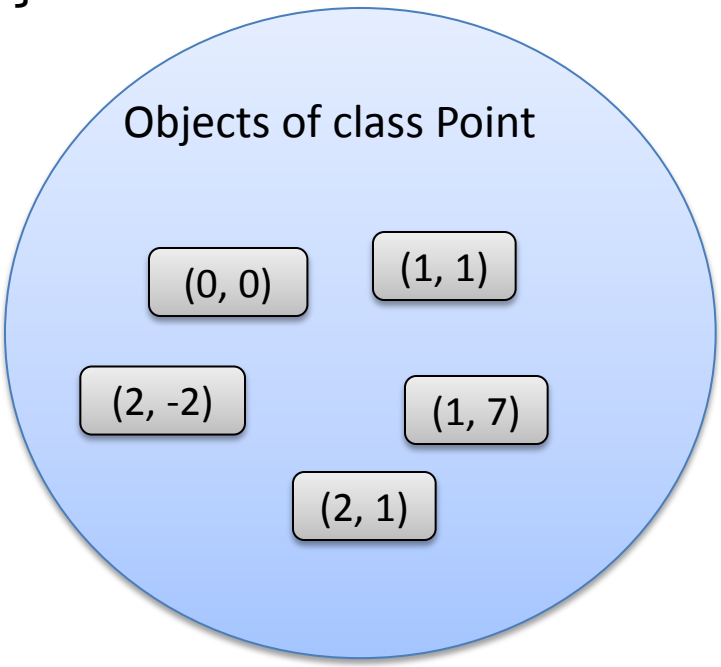
Objects (class instances)

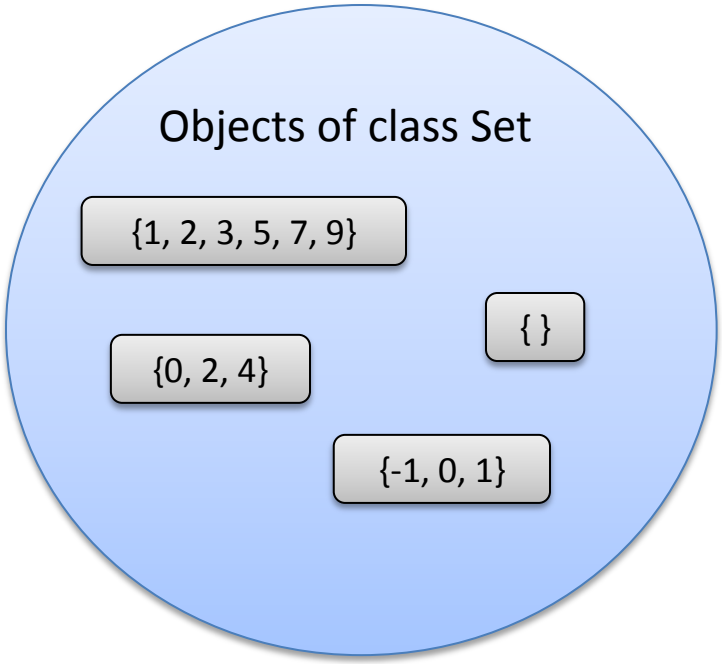
```
public class Point {
    int x;
    int y;
    ...
}
```

attributes

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

attributes





Construtors

- A constructor initializes the attributes of the objects of that class
  - There may be several constructors with different parameters
  - Constructors have no return type
  - Objects must be left in a valid state
  - Constructor is called using new

Good practice to define attributes as private.

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

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

   public Point(int x, int y) {
      this.x = x;
      this.y = y;
   }
}
```

this : used to disambiguate when parameter has the same vale as attribute

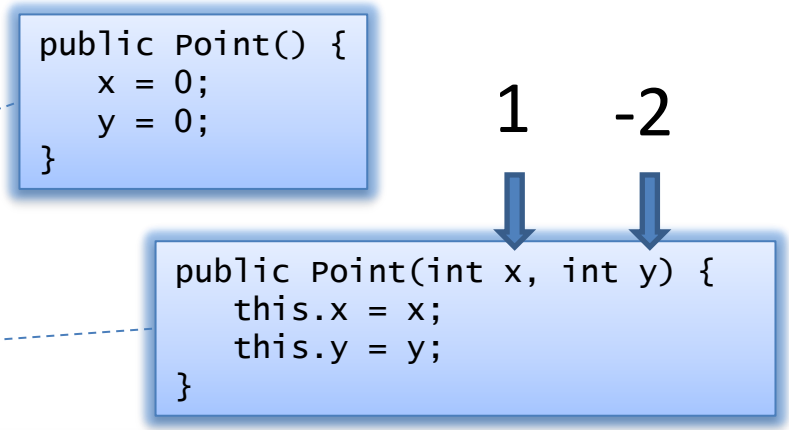
Object creation

- Operator new creates new objects

  `new` *`ClasseDoNovoObjecto`*`(`*`arguments`*`)`

- Arguments have to be compatible with one constructor
  - Examples:
    `new Point()`

    `new Point(1, -2)`



Automatic Initializations

- Attributes and elements of a vector of primitive types are initialized with default values:
  - int - `0`
  - double - `0.0`
  - boolean - `false`
  - …
- Reference attributes and elements of a vector are initialized to null

Instance methods

- Implementation of operations (object's behavior)
- Functions (calculate and return a result) or procedures (preform an *action*)
- May have parameters
- May change (modifiers) or not (inspectors) the object's state
- Generally functions are inspectors and procedures are modifiers

Inspectors

- Functions the return attribute values
- … or calculate values based on attributes
- Whether an inspector is necessary for each attribute is a context-dependant decision

```
public class IdCard {
   private String firstName;
   private String lastName;
   private int id;

   ...

   public String fullName() {
      return firstName + " " + lastName;
   }

   public int id() {
      return id;
   }

   ...
}
```

Could also be **getId()**

Modifiers

- When appropriate, a class may include methods to modify attribute values

```
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;
    }

    ...
}
```

id,never changes … should be final

Methods: functions and procedures

- Functions
  - Set of instructions with a well defined interface, that performs a calculation
  - Must return a result
  - Should not change the object's state

- Procedures
  - Set of instructions with a well defined interface, that perform an action (usually, change the object's state)
  - Do not return a value

Function

```
public class Name {
    private tipo attribute;
    ...

    public type name(parameters) {
        instructions
        return expression;
    }
}
```

Attributes should not be changed by function

signature

body

Procedure

```
public class Name {
    private type attribute;
    ...

    public void name(parameters) {
        instructions

    }
}
```

attributes may be modified by procedure

signature

body

Operations and methods in Java

- Operations
  - Part of the class interface
  - Callable
- Methods
  - Part of the class implementation
  - Executed when the corresponding operation is called
- One operation may be implemented by different methods

How? To see later …

Operations in Java: good practices

- Each operation must have a unique and well-defined objective (a function)
  - Inspectors – Name reflects what they return
  - Others – Name reflects the action they perform
- An operation should avoid to accumulate inspections and modification

Example: calculator

```
public class Calculator {
    private int value;

    public Calculator() {
        value = 0;
    }

    public int value() {
        return value;
    }

    public void set(int newValue) {
        value = newValue;
    }

    public void clear() {
        value = 0;
    }

    public void add(int term) {
        value = value + term;
    }
    ...
}
```

attribute

constructor

function

procedures

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

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

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

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

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

"Object calculator"

Encapsulation

- Encapsulation consists in “hidding” the attributes of an object from the outside (clients), separating *interface* and *implementation*, allowing:
  - More flexibility in the evolutionof the class implementation
  - More control over the correct use of class objects

Encapsulation

- All that can be private, should be!

- General rules
  - All attributes should be private
  - Constructors are usually public

Constants usually should be public.

Encapsulation

- Encapsulation may be applied using access modifiers:
  - public – allows direct external access
  - private – does not allow external access
  - … others to see later on

Attribute encapsulation

- Considered good practice

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

    ...
}

Rational r = new Rational(1, 4);

r.denominator = 0;
```

Compiler does not allow it

Separating interface and implementation

- Point in two dimensions (example)
  - One concept, two representations
    - Cartesian coordinates
    - Polar coordinates
  - Class to represent points
    - One interface, two (or more) implementations

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

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

Separating interface and implementation

```
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();
```

Separating interface and implementation

```
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();
```

Controlling object usage

```
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(..));
```

Program ends abruptly with error! (ArrayIndexOutOfBoundsException)

Controlling object usage

```
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(..));
```

Compiler identies error in external attribute access

More information/ References

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

Summary

- Classs and Objects
- Encapsulation