326.041 (2015S) – Practical Software Technology

(Praktische Softwaretechnologie)

Object Oriented Programming

Alexander Baumgartner
Alexander.Baumgartner@risc.jku.at

Research Institute for Symbolic Computation (RISC)
Johannes Kepler University, Linz, Austria

Non-Structured Programming

Earliest programming paradigm capable of creating Turing-complete (computationally universal) algorithms.

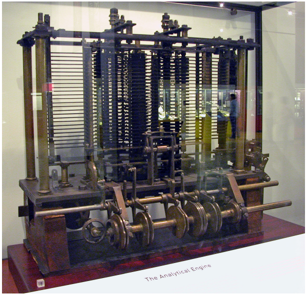


Figure: Analytical Engine, 1837, Charles Babbage

Non-Structured Programming

- Earliest programming paradigm capable of creating Turing-complete (computationally universal) algorithms.
- **Global data.**
- Only one main program.
- Program flow branching by command **GOTO.**

```
 ...
  50 IF X<>0 THEN GOTO 100
 ...
 100 PRINT X
 101 GOTO 25
 ...
```

Unstructured code (e.g. early BASIC).

Non-Structured Programming

- Edsger Dijkstra, 1968, Go To Statement Considered Harmful.
- William W. Cobern, Programming Language Choice: A Positive albeit Ambiguous Case for BASIC Programming in Secondary Science Teaching. He writes:
  - BASIC is not a structured language like Pascal and using it **fosters poor programming habits** that are very difficult to break,
  - there is **no** "ease of learning" **advantage** that would favor the use of BASIC over Pascal with introductory students.
- . . .

Block-Structured Programming

- **Global data.**
- Only one main program.
- Program flow is controlled by **program structures**.
  - if-then-else
  - while

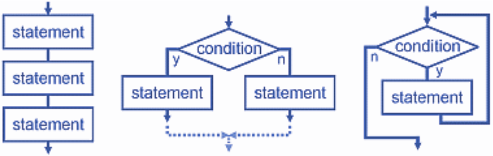


Figure: Program flow by structures.

```
  ...
  if x<>0 then begin
  ...
  end;
  else begin
  ...
  end;
  ...
```

Structured code (e.g. simple Pascal program).

Procedural Programming

- Program code is wrapped into **funcional substructures** (the procedures).
- **Data** is either **global** or **local to a particular procedure**.
- Data is passed among procedures as **arguments**.
- **Data structure definitions** are separated from the algorithmic program codes.
- Any given procedure might be called at any point during a program's execution, including by other procedures or itself.

```
int Fact(int n) {
    if (n = 0)
        return 1;
    return n * Fact(n - 1);
}
```

Procedural code – A procedur to compute the factorial.

Procedural Programming

- Poor modeling of the real world:
  - Procedures to carry out tasks.
  - Data (structures) to store information.
  - Real world objects might do both.
- E.g. A **thermostat** control program:
  - 2 procedures: **heating_on()** and **heating_off()**.
  - 2 global variables: **currentTemp** and **desiredTemp**.
- Crude organizational units:
  - The above procedures and variables do not form a programming unit, which you could call thermostat.

Modular programming

- Separating the functionality of a program into independent, interchangeable modules.
- Algorithms and their dependent data are wrapped into modules.
- The interfaces of the modules are well defined.
- E.g.: Modula, Java, Haskell,. . .

Independent paradigms

- **Object oriented programming.**
- Functional programming.
- Logic programming.
- Literate programming.
- . . .

Early OO Languages

- **Simula 67:** First OO language. By Dahl and Nygaard in the 60s.
  - Derived from Algo 60.
  - Uses classes and inheritance.
  - Methods/behaviors have not been bound strictly to the objects yet.
- **Smalltalk:** First consequent OO language. By Kay et al. in the 70s.
  - Influenced by Simula.
  - Everything is an object.
  - Already a development tool with GUI.
  - Is still used at present.
  - It had a strong influence for many other OO languages.

Definition of OO?

There is **no accurate definition** which is accepted by everyone.

- Nygaard (1926-2002), one of the developers of Simula 67, says:
    - A program execution is regarded as a physical model simulating the behavior of either a real or imaginary part of the world.
- Kay, one of the developers of Smalltalk, requires the following essential elements for an OO language:
    - Polymorphism.
    - Data encapsulation.
    - Inheritance.
    - Every type is an object type.
    - The object types compose a hierarchy with a single root.

Objects

Grady Booch. Object-Oriented Analysis and Design with Applications:

An Object has state, behavior and identity.

- State = Data.
- Behavior = Algorithms which use the data.
- Identity = Distinguishably from other objects.

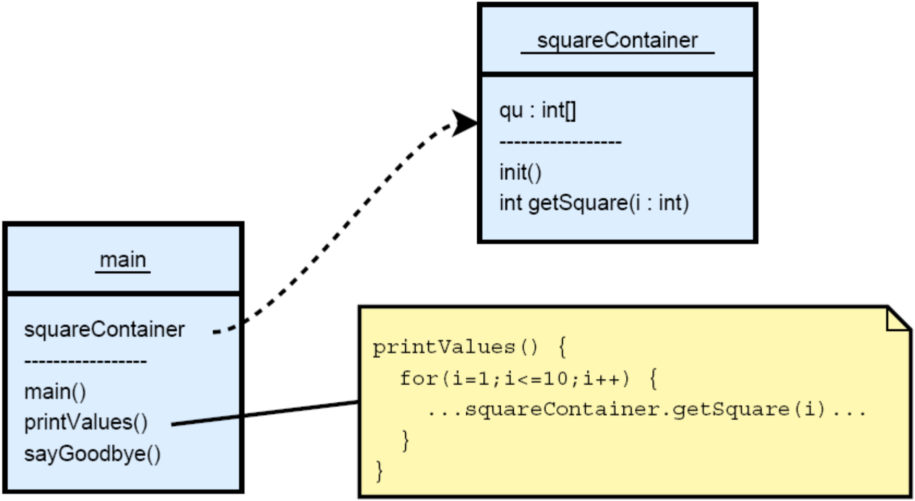
Object Based Programming

- The global state of a program consists of (the states of) numerous objects.
- Objects interact with each other via messages.
- Messages are realized as procedure/method calls, e.g.:.
  - sending message “m” to object “o” =
    calling procedure “m” of object “o”.
  - Procedure “m” is able to modify directly the state of the objects “o” or to send another message to another object.

Object-Oriented Design

- **Abstraction**:
  - Distill a complicated system down to its most fundamental parts.
  - Describing parts of a system by naming them and explaining their functionality. (In Java: Interfaces and abstract classes.)
  - Forces encapsulation and enables modularity.
  - **Flexibility & Adaptability**: Implementations are interchangeable.
- **Modularity**:
  - Programs are divided into separate functional units.
  - **Robustness**: Test and debug separate components before integrating them into a larger software system.
  - **Reusability**: Same components are used in several software system.
- **Encapsulation**:
  - Components should not reveal implementation details.
  - **Robustness & Adaptability**: Allows changing implementation without adversely affecting other parts $\Longrightarrow$ Fix bugs, improve implementation (e.g. performance), or add new functionality by local changes.

Example for Objects



```
printValues() {
  for(i=1;i<=10;i++) {
    ...squareContainer.getSquare(i)...
  }
}
```

Figure: The two objects **main** and **squareContainer**.

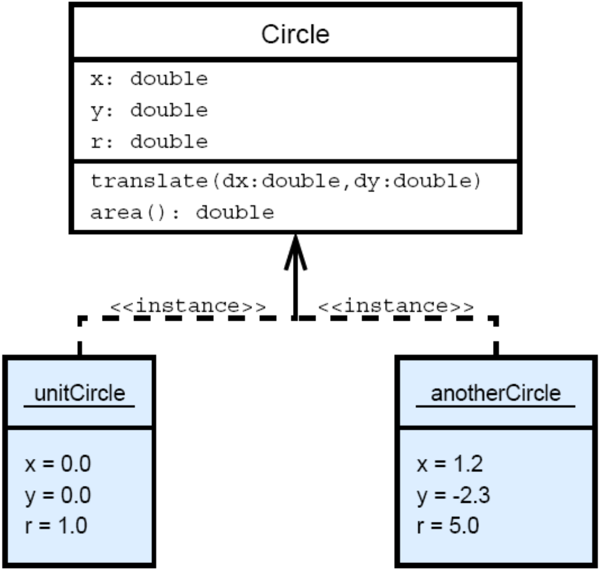
Encapsulation

- Accessing to the field “squareContainer.qu” from outside (e.g.: from method “main”) is not possible/desirable.
- Accessing (changing/reading values) to the fields of an object is done typically though designated access points (public methods).
- **Advantages:**
  - avoiding side effects,
  - clear structures (storing the data and their algorithms together),
  - controlling the modification of the data, etc.

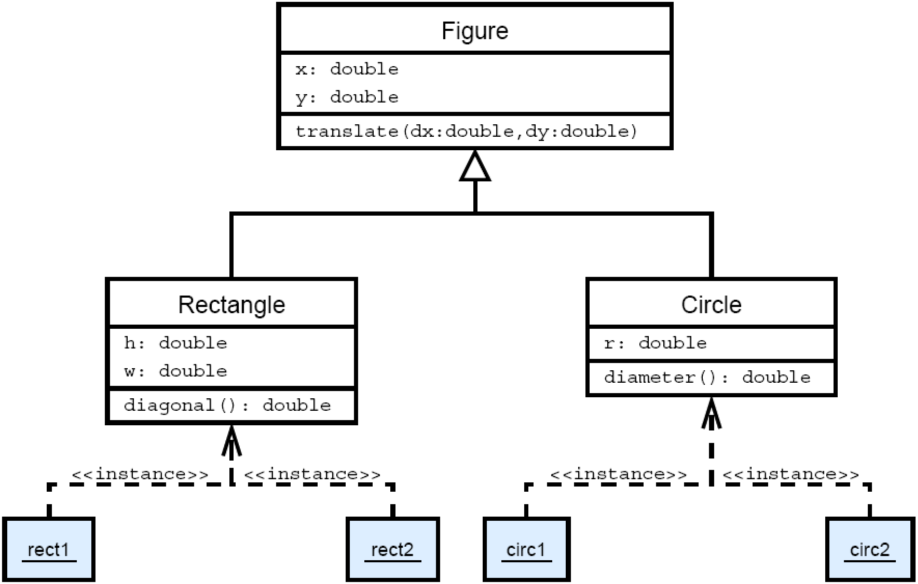
Object Oriented Programming

- **Typing:** Objects belong to classes. Within a class each object has
  - the same data fields and
  - the same behavior (same methods).
- **Inheritance:** A class may inherit the data and behavior of (an)other class(es).
- **Polymorphism:** The same piece of program/function can work on different kind of objects.

Example for Class and Objects



Example for inheritance



Example for Polymorphism

```
public static void main(String[] args) {
    Figure f;
    Rectangle r = ...;
    Circle c = ...;

    f = r;  // Allowed, Rectangle is subclass of Figure
    f = c;  // Allowed, Circle is subclass of Figure

    r = c;  // Not allowed
    r = f;  // Not allowed
}
```

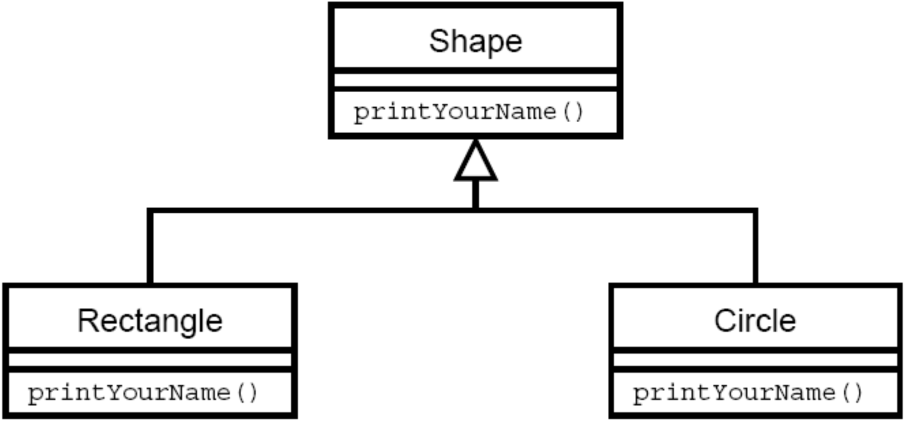
Polymorphism – Implicit upcasting.

Influence of Polymorphism

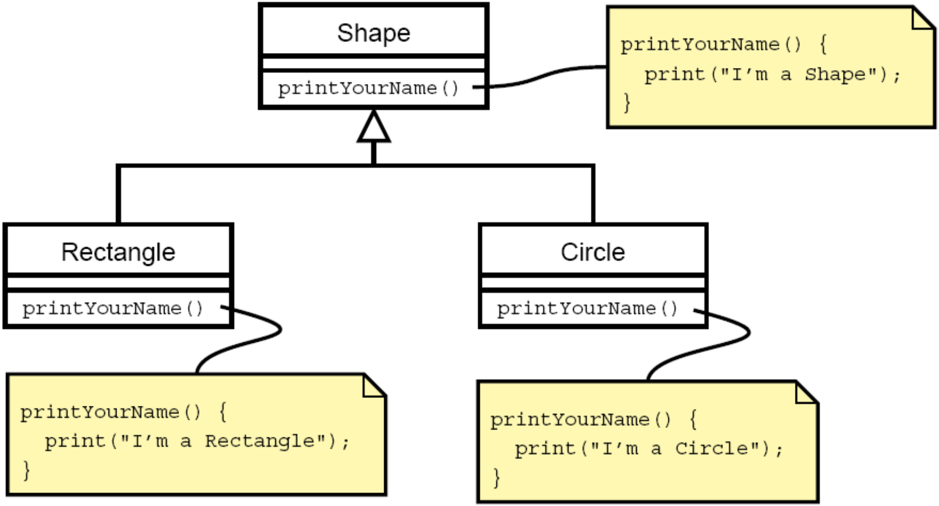
```
public static void main(String[] args) {
    Rectangle r = ...;
    Circle c = ...;

    diagTranslate(r, 1.0);
    diagTranslate(c, 2.0);
}

public static void diagTranslate(Figure f, double d) {
    f.translate(d, d);
}
```

Diagonal translate a Figure.

Dynamic/Late Binding I



Dynamic/Late Binding I

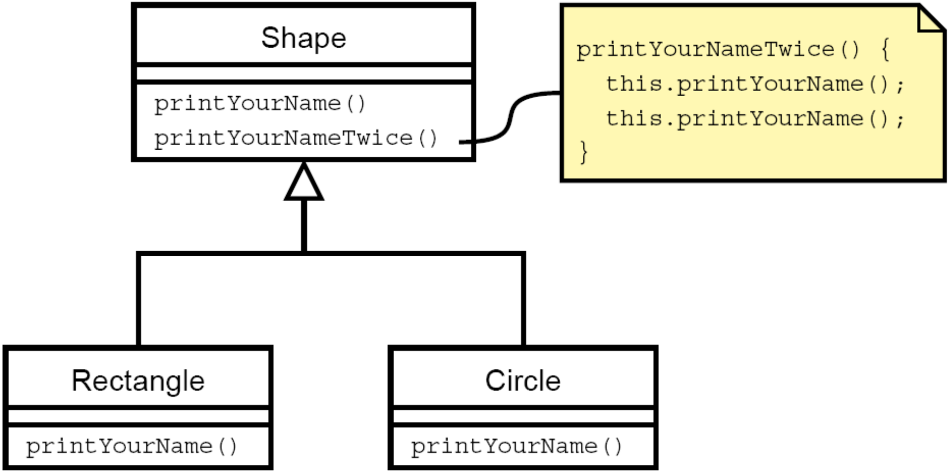


```
printYourName() {
  print("I'm a Shape");
}
```

```
printYourName() {
  print("I'm a Rectangle");
}
```

```
printYourName() {
  print("I'm a Circle");
}
```

Dynamic/Late Binding II



```
printYourNameTwice() {
  this.printYourName();
  this.printYourName();
}
```