Object Oriented Programming and Design

UD

Jul’2022

Classes and Objects

- A *class* is a prototype for creating objects

- A class has a *constructor* for creating objects

A Simple Class, called "Time"

```
class Time {

    private int hour, minute;

    public Time (int h, int m) {
        hour = h;
        minute = m;
    }

    public void addMinutes (int m) {
        int totalMinutes = ((60*hour) + minute + m) % (24*60);
        if(totalMinutes<0)
            totalMinutes = totalMinutes + (24*60);
        hour = totalMinutes / 60;
        minute = totalMinutes % 60;
    }

}
```

constructor for Time

C Example: Represent a Time

```
struct TimeTYPE {
        short Hour;
        short Minute;
}
struct TimeTYPE inToWork, outFromWork;
   ::
inToWork.Hour = 9;
inToWork.Minute = 30;

outFromWork.Hour = 19;
outFromWork.Minute = 35;
```

Java Example: Represent a Time

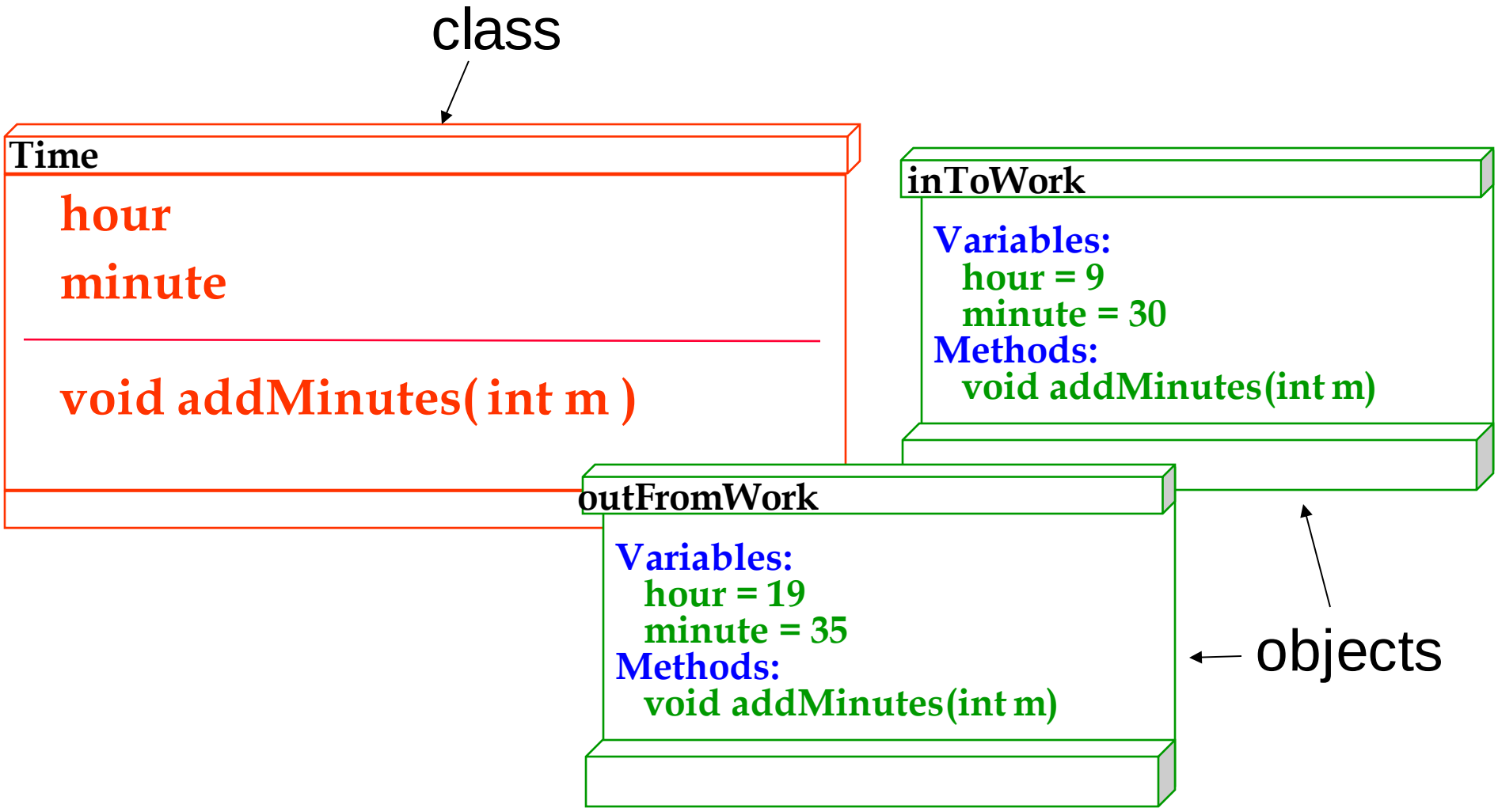
```
class Time {

private int hour, minute;

public Time (int h, int m) {
     hour = h;
     minute = m;
}
…
}
```

attributes of Time

constructor for Time

```
Time inToWork = new Time(9, 30);
Time outFromWork = new Time(19, 35);
```

Objects


class
Time
hour
minute
void addMinutes(int m)
inToWork
Variables:
hour = 9
minute = 30
Methods:
void addMinutes(int m)
outFromWork
Variables:
hour = 19
minute = 35
Methods:
void addMinutes(int m)
objects

Structure of a Class Definition

```
class name {

declarations


constructor definition(s)


method definitions

}
```

← Members / attributes and symbolic constants

← how to create and initialize objects

← how to manipulate the state of objects

These parts of a class can actually be in any order

History of Object-Oriented Programming

- SIMULA I (1962-65) and Simula 67 (1967) were the first two object-oriented languages
  - Developed at the Norwegian Computing Center, Oslo, Norway by Ole-Johan Dahl and Kristen Nygaard
  - Simula 67 introduced most of the key concepts of object-oriented programming: objects and classes, subclasses ("inheritance"), virtual procedures

The Ideas Spread

- Alan Kay, Adele Goldberg and colleagues at Xerox PARC extend the ideas of Simula in developing **Smalltalk** (1970's)
  - Kay coins the term "object oriented"
  - Smalltalk is first fully object oriented language
  - Grasps that this is a new programming paradigm
  - Integration of graphical user interfaces and interactive program execution
- Bjarne Stroustrup develops **C++** (1980's)
  - Brings object oriented concepts into the C programming language

Other Object Oriented Languages

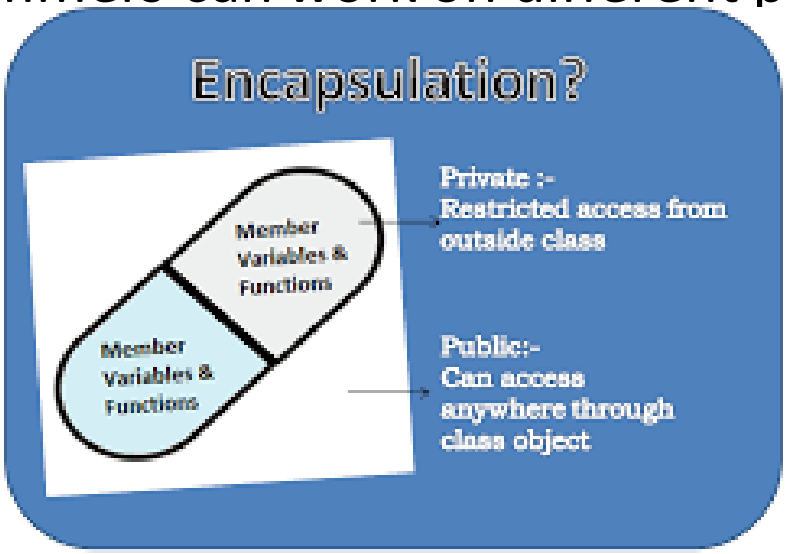
- Eiffel (B. Meyer)
- CLOS (D. Bobrow, G. Kiczales)
- SELF (D. Ungar et al.)
- Java (J. Gosling et al. at Sun Microsystems'1991; initially this language was called *Oak* and renamed as Java in '1995)
- BETA (B. Bruun-Kristensen, O. Lehrmann Madsen, B. Møller-Pedersen, K. Nygaard)
- Other languages add object dialects, such as TurboPascal
- …

Features of OOP

- Encapsulation
- Object Identity
- Polymorphism
- Inheritance

Encapsulation

- Mechanism for binding together code and data and providing controlled access
- Code, data or both may be declared as *private* or *public*
- *private* code or data accessible only by another part of the object and not outside; *public* can be accessed also outside that object
- Java's basic unit of encapsulation is *class*; defines the form or prototype of *object*
- Different groups of programmers can work on different parts of the project



Advantages of Encapsulation

Building the system as a group of interacting objects:

- Allows extreme modularity between pieces of the system
- May better match the way we (humans) think about the problem
- Avoids recoding, increases code-reuse
- Safe from outside interference and misuse

Object identity

- Each created object has unique identity (memory address). e.g.

  Student s1 = new Student("Joy");

  Student s2 = s1;

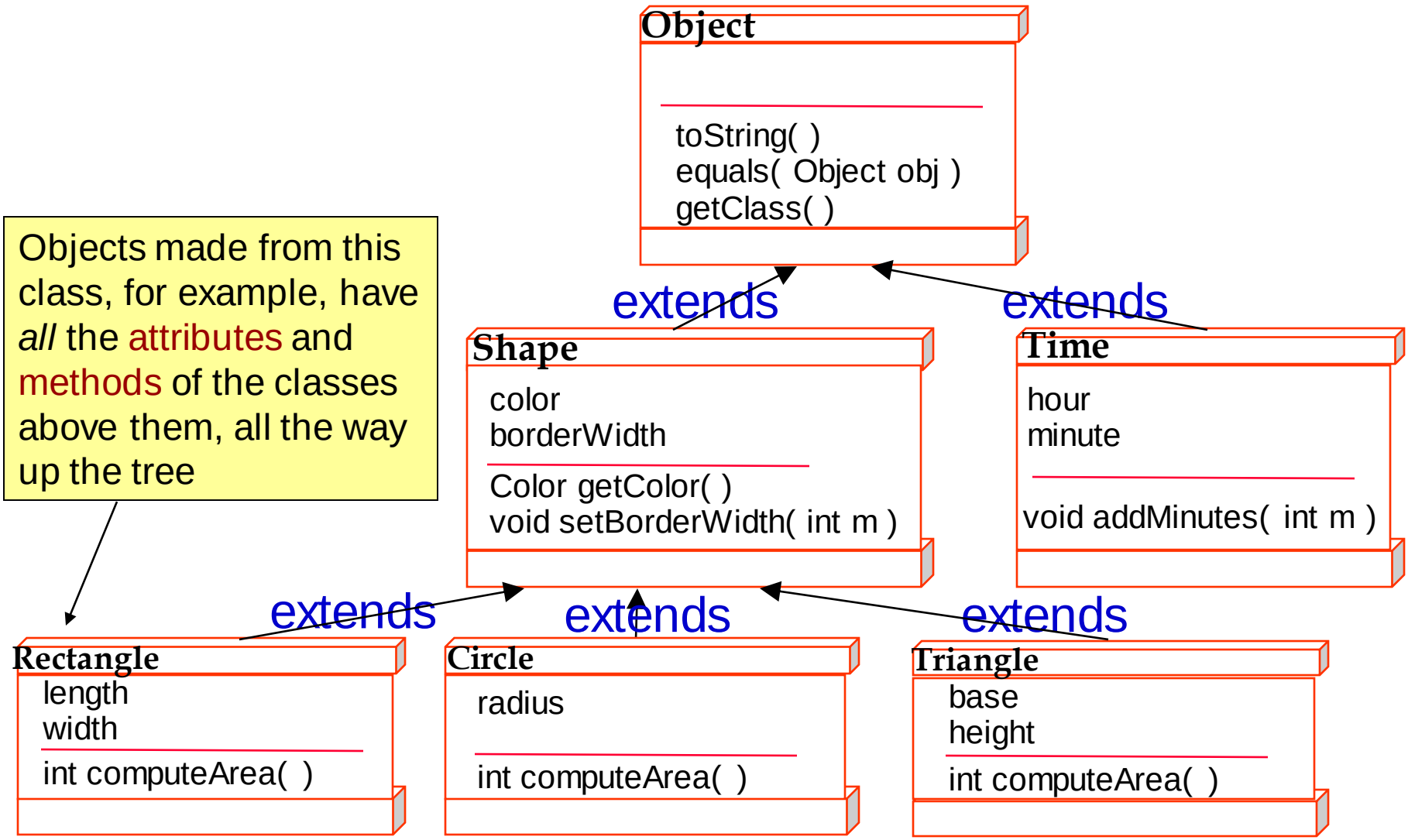
  Student s3 = new Student("Joy");

Note: s1 and s3 separate identities though with same state (value). Guess values of (s1 == s3), (s1 == s2)

Inheritance

- Process by which one object can acquire properties of another object
- Classes can be arranged in a hierarchy
- Subclasses *inherit* attributes and methods from their parent classes
- This allows us to organize classes, and to avoid rewriting code – new classes *extend* old classes, with little extra work!
- Allows for large, structured definitions

Example of Class Inheritance

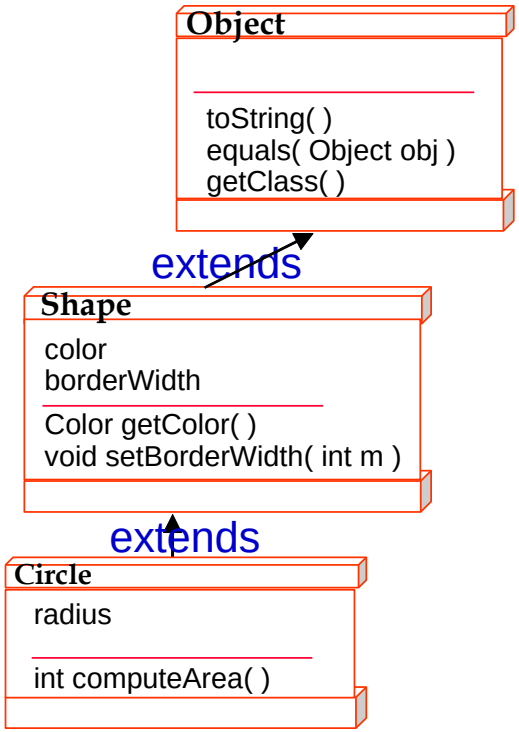


Polymorphism

- An object has “multiple identities”, based on its class inheritance tree
- It can be used in different ways, e.g.

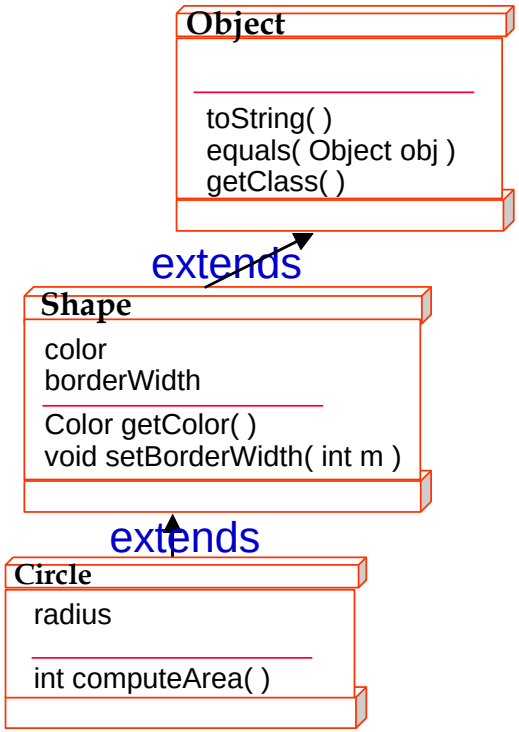
Polymorphism

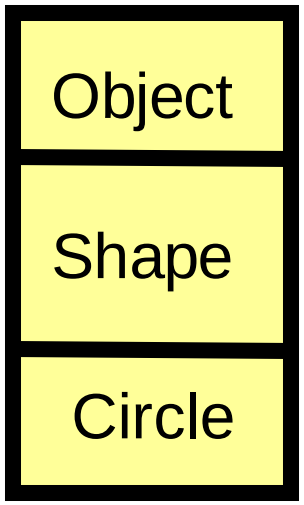
- An object has “multiple identities”, based on its class inheritance tree
- It can be used in different ways
- A Circle is-a Shape is-a Object



Polymorphism

- An object has “multiple identities”, based on its class inheritance tree
- It can be used in different ways
- A Circle is-a Shape is-a Object

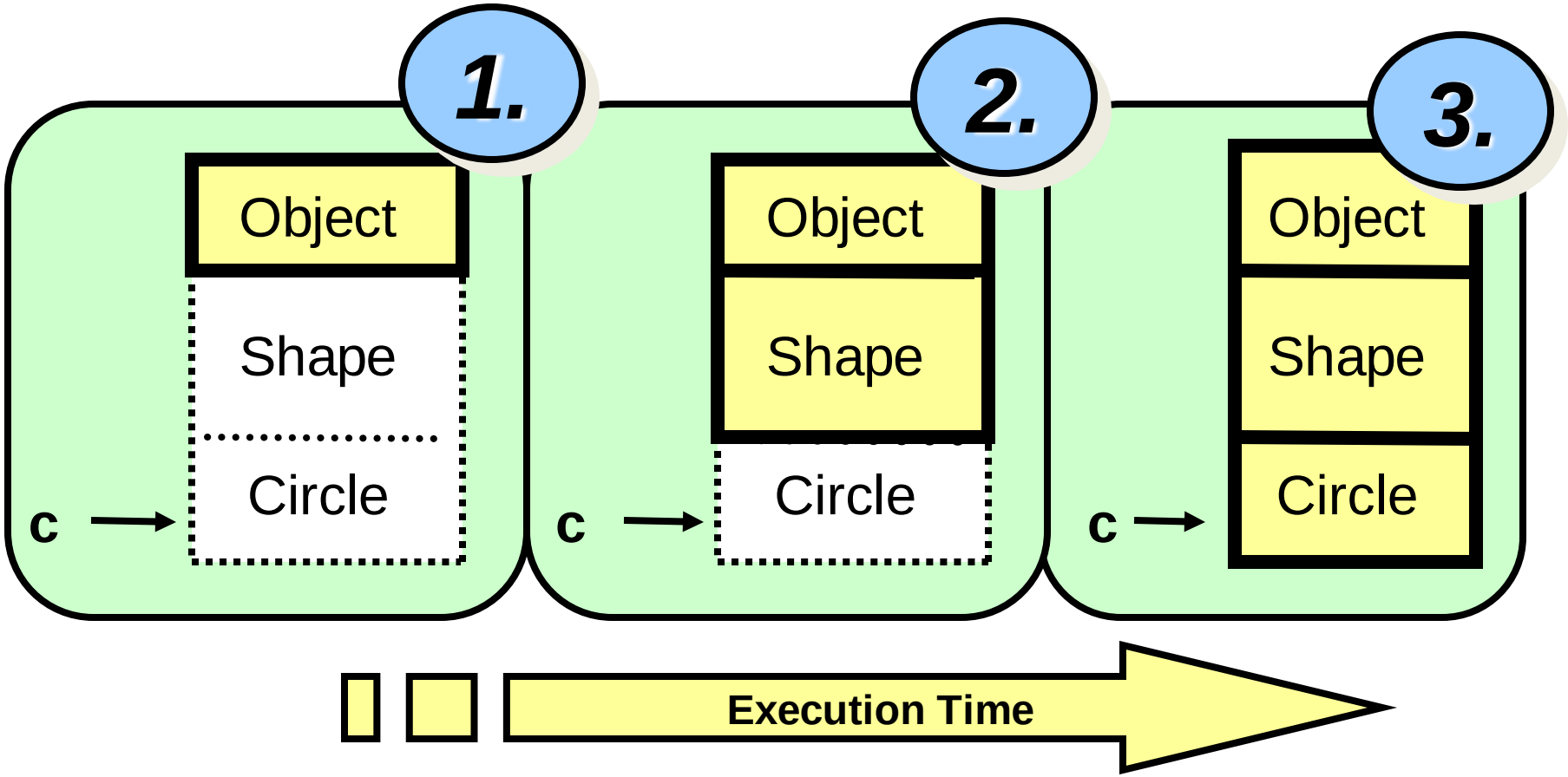




A Circle object really has 3 parts

How Objects are Created

```
Circle c = new Circle( );
```

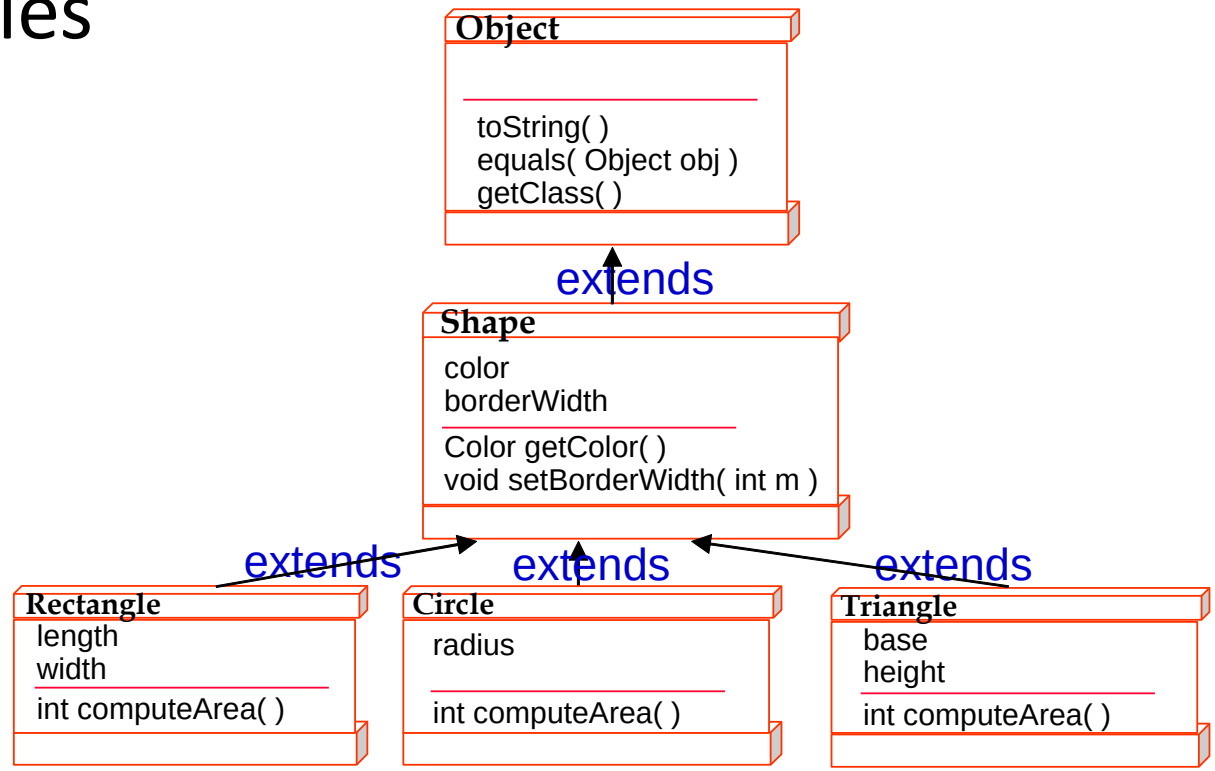


Three Common Uses for Polymorphism

1. Using Polymorphism in Arrays
2. Using Polymorphism for Method Arguments
3. Using Polymorphism for Method Return Type

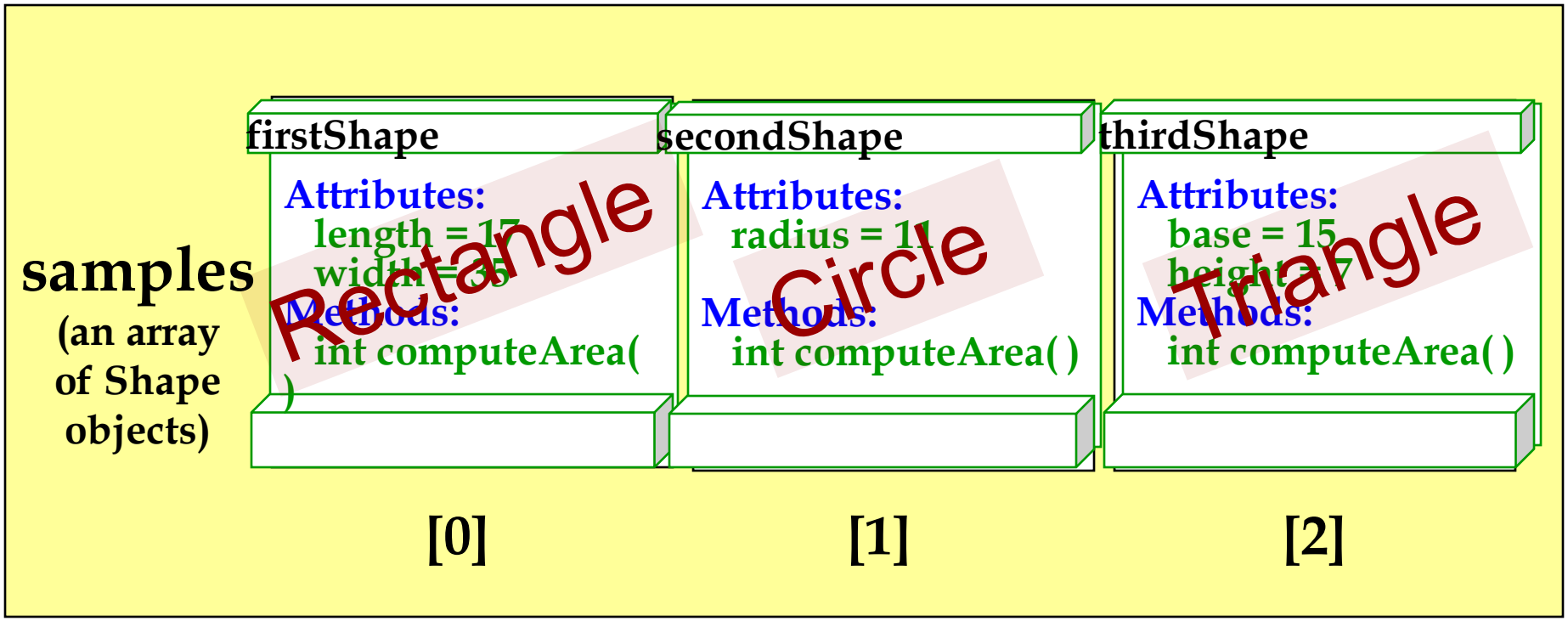
1) Using Polymorphism in Arrays

- We can declare an array to be filled with “Shape” objects, then put in Rectangles, Circles, or Triangles



1) Using Polymorphism in Arrays

- We can declare an array to be filled with “Shape” objects, then put in Rectangles, Circles, or Triangles



2) Using Polymorphism for Method Arguments

- We can create a procedure that has Shape as the type of its argument, then use it for objects of type Rectangle, Circle, and Triangle

```
public int calculatePaint (Shape myFigure) {
    final int PRICE = 5;

    int totalCost = PRICE * myFigure.computeArea( );
    return totalCost;
}
```

Actual definition of computeArea( ) is known only at runtime, not compile time – this is "dynamic binding"

OOP: Major & Minor Elements

- Major Elements

(i) Abstraction, (ii) Encapsulation, (iii) Modularity, (iv) Hierarchy

  - Not OOP if any of these major elements are absent

- Minor elements

(i) Typing, (ii) Concurrency, (iii) Persistence

  - Useful but not indispensable for OOP

Abstraction

- Denotes the essential characteristics of an object that distinguish it from all other objects

While designing class Student, following attributes typically included:

- roll_number
- Name
- course
- address…

…while following characteristics are eliminated:

- pulse_rate
- colour_of_hair..

Encapsulation & Modularity

- Discussed over previous slides.
- Questions?

Hierarchy

- Ranking or ordering of abstraction
- Using this, a system can be made up of interrelated subsystems and levels of subsystems until the smallest level components
- Uses “Divide and conquer” principle
- Types: IS-A and PART-OF

If we derive a class Rose from a class Flower, we can say that a rose “is–a” flower.

A flower is composed of sepals, petals, stamens, and carpel. It can be said that a petal is a “part–of” flower.

Typing

- Characterization of a set of elements, e.g. a class is a type having properties distinct from any other type

- Categories:
  - Strong Typing: Object' operation checked at the time of compilation, e.g. Java, Python.
  - Weak Typing: Object' operation checked at the time of execution, e.g. C, JavaScript.

Concurrency

- Concurrency (in operating systems) allows performing multiple tasks or processes simultaneously:
  - Single thread of control: When a single process exists in the system
  - Multi thread: Most systems with multiple threads - some active, some waiting for CPU, some suspended and some terminated. Available on multi-CPU or even on single-CPU

Persistence

- An object occupies a memory space and exists for a particular period of time
- Lifespan of an object is typically the lifespan of the execution of creator program
- In files or databases, the object lifespan is longer than the duration of the creator process.
- Thru persistence, an object continues to exist even after its creator ceases to exist

Links and Association

- **Link**: Connection thru which an object collaborates with other objects, i.e. one object may invoke methods in another object
  - Depicts relationship between 2 or more objects

- **Association**: Group of links having common structure and common behavior.
  - Depicts relationship between one or more classes
  - Link defined as an instance of an association

e.g., Student and Faculty having association

Degree & Cardinality

- **Degree of Association**
  - Unary (connects objects of same class)
  - Binary or Ternary

- **Cardinality of Binary Association**
  - One-to-one
    - A single object of class A is associated with a single object of class B, e.g. Employee and Project
  - One-to-many, e.g. College and Student
  - Many-to-many, e.g. Faculty and Subject

Aggregation

- Relationship among classes by which a class can be made with any combination of objects of other classes.

- Allows objects to be placed directly within the body of other classes
  - Referred as a “part–of” or “has–a” relationship, with the ability to navigate from the whole to its parts.

- An aggregate object is an object composed of one or more other objects

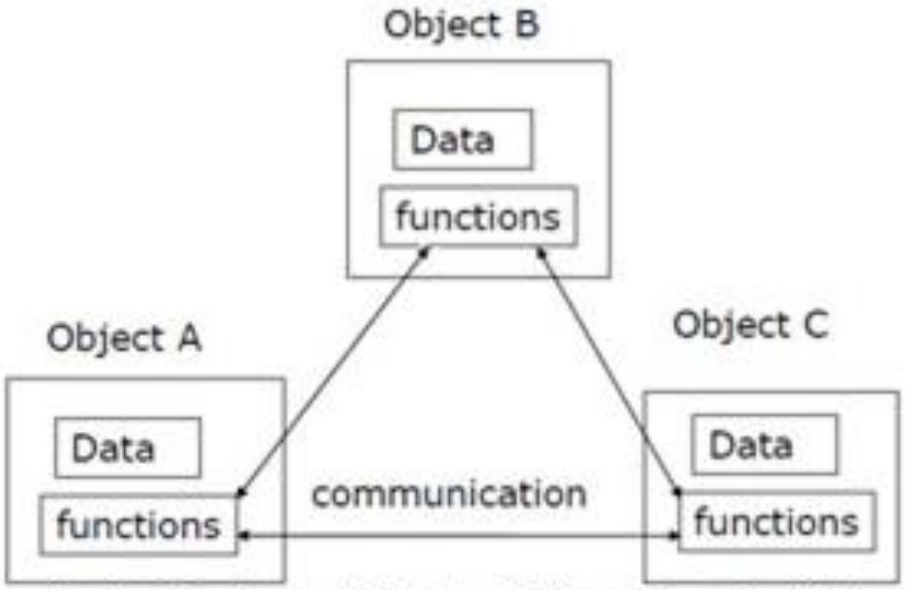
- **Composition**: Aggregation in restricted form, e.g.
  - Library and Students: having aggregation relationship
  - Libray and Books: having composition relationship

Aggregation (contd.) & Meta-class

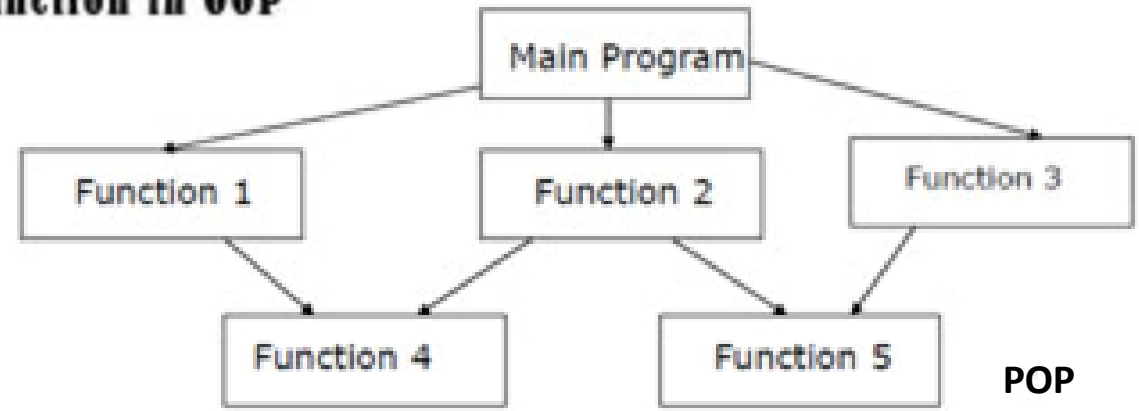
- In the relationship “a flower has–a petal”, flower is the whole object or the aggregate and petal is a “part–of” the flower
- Aggregation may denote:
  - **Physical containment** : e.g., a computer is composed of monitor, CPU, mouse, keyboard etc.
  - **Conceptual containment** : e.g., shareholder has–a share

Meta-class: Class of a class, i.e. class whose instances are classes, e.g. directly supported in Small talk, Python etc.

POP vs. OOP



Organization of Data & Function in OOP



POP

POP vs. OOP (contd.)

#	Category	Procedure Oriented Programming	Object Oriented Programming
1.	**Priority**	Importance given to the sequence of functions to be done i.e. algorithms	Importance given to the data
2.	**Program Division**	Larger programs divided into functions	Larger programs divided into objects
3.	**Data Sharing**	Most functions share global data i.e. data move freely around the system from function to function.	Mostly the data is private and only functions inside the object can access the data.
4.	**Approach**	Follows a top down approach in problem solving	Follows a bottom up approach
5.	**Code Extensibility**	Adding of data and function is difficult, as full program may need to be changed	Adding of data and function is easier, and full program is not required to be changed
6.	**Access Specifiers**	There is no access specifier	There is public, private, default and protected specifiers

POP vs. OOP (contd.)

#	Category	POP	OOP
7.	**Communication across modules**	Procedure / function connect with each other by parameter passing	Objects communicate via message passing
8.	**Content of module**	Every function contains different data	Data & associated functions of each individual object act like a single unit
9.	**Focus**	Functions with sequence get more importance than data	Data gets more importance than functions
10	**Data hiding**	There is no perfect way for data hiding	Data hiding possible which prevent illegal access of functions from outside
11	**Overloading**	Operator cannot be overloaded.	Can be overloaded in OOP languages like C++ (not possible in Java)
12	**Security**	Less in the absence of data hiding.	More as prevention of illegal access is possible
13	**Example**	Pascal, FORTRAN, C	C++, Java, Small talk, Python etc.

Advantages of OOP

- Improved software-development productivity ← extreme modularity, easy extensibility & reusability of objects
- Improved software maintainability ← part of the system can be changed w/o affecting large-scale
- Faster development ← Reusability of code & rich libraries of objects
- Lower cost of development ← Reuse of software (more effort into OOAD)
- Higher-quality software ← Above 2 points allow more time & resources in the V&V

Disadvantages of OOP

- Steep learning curve: Complex to create programs based on objects' interaction. Initial challenges in conceptualizing Inheritance & Polymorphism..
- Larger program size: Typically more lines of code..
- Slower programs: Typically slower as they typically require more instructions to be executed
- Not suitable for all types of problems: Some problems can be solved more efficiently with functional-programming style, logic-programming style, or procedure-based programming style..

Why Java for OOP?

- Java is Object-Oriented (*syntax inherited from C and object model adapted from C++*)
- Designed with lessons learnt and best experiences from other OO languages used for many years
- Java has strong type checking
- Java handles its own memory allocation
- Java's syntax is "standard" (similar to C and C++)
- More…
  - Portability and Security via Java Bytecode
  - Web based programming

Object-Oriented Programming in Industry

- Large projects are routinely programmed using object-oriented languages..

- **MS-Windows** and applications in **MS-Office** – all developed using object-oriented languages