ECE 462
Object-Oriented Programming using C++ and Java

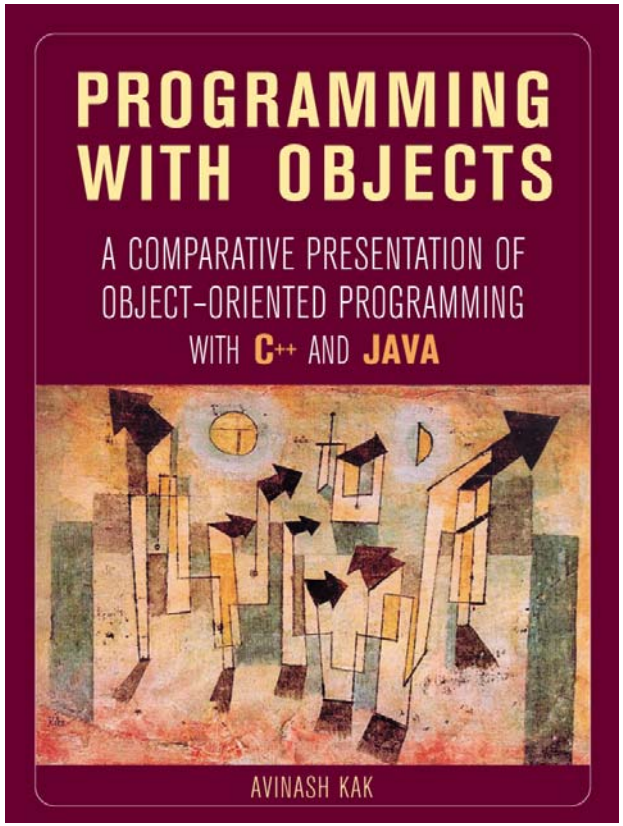
Lecture 1

Yung-Hsiang Lu

yunglu@purdue.edu

Textbook

- "Programming with Objects" by Kak, John-Wiley
- source code from the book and errata: http://programming-with-objects.com/
- many executable examples about the concepts explained in the book
- contains topics that are rarely discussed in other books, such as multiple inheritance
- provides frequent comparison between C++ and Java



Experiment:
Directed Problem Solving in Labs

- This semester, we are conducting a pedagogical experiment, similar to DPS in ECE270 and 362.
- All lectures are recorded and available on-line.
- Every week = 2 lectures + 1 lab
- There is **no reduction** of the course material because QA is handled outside lectures and no time is wasted setting up computer or demonstration of tools.
- Lab sessions = office hours. Twelve (12) lab assignments are graded. Additional office hours are also available.
- You can watch the lecture videos at any time. You are encouraged to watch the videos with classmates, pause, and discuss.

Advantages of DPS

- Determine your own pace. You can watch is once or multiple times (asynchronous learning)
- Encourage group study by watch the video together.
- Promote self assessment, using the time you need.
- Provide more flexibility in utilizing your time.
- Enhance student-instructor interaction in labs
- Accommodate the wide range of students' background
- Offer one-to-one attention to individual's learning needs during the lab hours

Disadvantage of DPS

- no classroom interaction with classmates $\Rightarrow$ watch lecture videos with classmates
- no classroom interaction with instructor $\Rightarrow$ use lab hours
- cannot ask questions during lectures $\Rightarrow$ post questions in webct

- **Please provide frequent feedback and suggestions through webct discussion.**
- **Demonstration of webct video and discussion.**

Grading

- 1% course evaluation (send email to instructor after completion)
- 1% “technology and education” survey and discussion, organized by Professor Brown (in class, November)
- 4% homework = 0.5% x 8, in webct
- 12% lab assignments = 1% x 12
- 20% programming assignments = 4% x 5
- 10% exam 1 (in class, September 21, open book, no collaboration)
- 14% exam 2 (in class, October 19, open book, no collaboration)
- 18% exam 3 (in class, November 16, open book, no collaboration)
- 20% final exam (TBD, open book, no collaboration)
- > 85% A. 75% -85% B. 65% - 75% C ... after normalization by the highest score (if < 100%) in class
- Each outcome is tested twice. Failing one or multiple outcomes in both tests $\Rightarrow$ F

Bonus Points

- Each person can receive up to 5 bonus points:
  - up to 4 points, provide 5 or more questions for each exam (posted in webct), one point per exam
  - up to 5 points, scribe the lectures (every word) and post it in webct, one point per lecture
  - 1 point: the two people that have the most postings in webct discussion before final exam
- No other bonus point will be given.

Lab Assignments

- Each person must submit 12 lab assignments. Only one submission can be accept per session.
- Each assignment asks you to learn one programming tool and must be signed off by the teaching assistant.
- You are welcome to use the lab hours even after you have submitted all assignments.
- The instructor and the teaching assistant will be available in the lab sessions as office hours. Additional office hours can be arranged by appointment.
- You can discuss lab assignments with anyone.

Programming Assignments

- 3 regular programming assignments: 2 in Java and 1 in C++, each 4% of the grade.
- 1 programming assignment either Java or C++, divided into 2 stages: planning and implementation, each 4%.
- You can do each assignment alone or work with one classmate. You **may change** the group mate for each assignment.
- You can discuss programming assignments with anyone but you are allowed to **share code only with your group mate**. In this course, **all deadlines are firm** (no extension and no exception for any reason, including but not limited to earthquake, tsunami, tornado, invasion by outer space aliens, power failure, fire, flood).
- Each person has one 24-hour "free" late day. If you have a group mate, you two can have 2 free late days, use 2 days once, or one day twice.

Programming Assignment 4

- You (and your group mate) decide what to do. You can choose Java or C++ or both.
- Requirements:
  - object-oriented
  - graphical user interfaces + networking
  - UML diagrams of **all** classes, at least 5 use cases, at least 5 sequence diagrams, at least 1 state diagram
  - schedule + testing plan and results
- Submit a detailed plan on November 9.
- Submit the program + documentation on November 30.

Exams
(open book, open note, individual)

- multiple choice, short answer (code statement), short code (several lines)
- Final exam may contain several questions of slightly longer (about 10-20 lines) of code
- **"zero-tolerance" of dishonesty**: violations will be reported to the associate head of ECE, no exception. We will use similarity checking in your assignments.
- You can discuss lecture, homework, lab, or programming assignments with anyone. You can share code with **only** your programming partner (if you have one).
- Regarding must be submitted by a written request (or email) within one week. You are **not allowed** to ask or discuss with TA about regrading.

Prerequisites

- ECE 264
- Know how to write and compile C programs in UNIX-based (e.g. Linux or Solaris) machines, such as gcc, gdb, and Makefile
- Understand the concept of pointers in C
- We will **not** emphasize **syntax**. Instead, we will spend more time on how to design and implement non-trivial programs.

Objects

- Object: a “concrete and tangible” entity that can be separated with unique properties. Examples: you, your book, your car, my computer, Tom, Amy’s computer, a window on your computer desktop, your phone, Sam’s digital camera, Jennifer’s pager ...
- Object can be "abstract": a triangle, a database, a browser ...
- Each object is unique and can be **identified** using name, serial number, relationship with another object ...
- Each object has a set of **states**, such as location, speed, size, address, phone number, on/off ...
- Each object has unique **behavior**, such as ring (phone), accelerate and move (car), resize (window), take picture (camera), send email (computer), display caller (pager)

Objects' Properties

- Each object has **three** important properties:
  - unique identity
  - states (also called attributes), noun
  - behavior (action), verb
- Objects can interact, for example
  - You (object) press (action) the pedal (object) of your car (object). As a result, your car accelerate (action).
  - When your phone (object) rings (action) and alerts (action) you (object) of an incoming call (state), you answer (action) the call (state).
  - You submit (action) homework (object) and it is graded (action) with a score (state).

What is Class?

- A class provides
  - a type (similar to “struct”) to create objects
  - an interface (methods) for objects to interact (“send messages”), such as setLineStyle and getArea
  - a set of attributes
  - implementation of the interfaces
  - a base for derived classes
- Review: objects- concrete entities, such as John, your car, my book, the phone on this desk... class- a representation of the commonality of objects, such as Human, MotorVehicle, MobilePhone...

Self Test

- We have many self tests. Pause the video and resume when you are ready.
- In the following list, mark C as class and O as object.

- your home
- MSEE
- LaptopComputer
- his wallet
- the bottle of water I drank yesterday
- my favorite restaurant
- the McDonald across the street

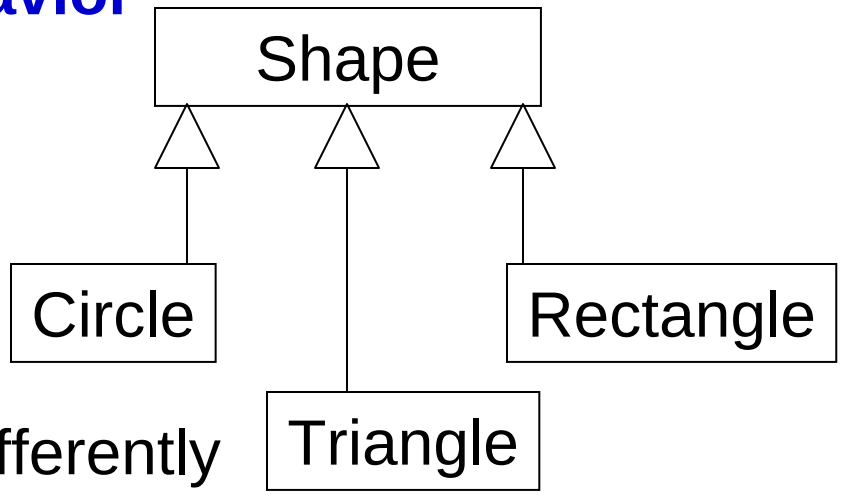
- Building
- Tom’s PDA
- Wallet
- Water
- BottleWater
- Restaurant
- FastFoodRestaurant

Self Test Answers

- your home (O)
- MSEE (O)
- LaptopComputer (C)
- his wallet (O)
- the bottle of water I drank yesterday (O)
- my favorite restaurant (O)
- the McDonald across the street (O)
- Building (C)
- Tom’s PDA (O)
- Wallet (C)
- Water (C)
- BottleWater (C)
- Restaurant (C)
- FastFoodRestaurant (C), may be a derived class of Restaurant

Objects and Classes

- Organize objects with similar properties (states and behavior) into classes: human, car, computer, phone, window, circle, rectangle, triangle, bridge, skyscraper ...
- Inheritance: find **commonality** among **classes** (not objects) and create a base class that represents the common **interfaces** and **behavior**
- Common interface of Shape:
  - center
  - line style and thickness
  - area ...
  - But areas are computed differently



Inheritance
to achieve better code reuse

```
class Shape {
   public void setColor(Color inColor);
   public void setLineStyle(LineStyle lineStyle);
   public void setLineThickness(int thickness);
}
class Circle extends Shape {      /* Circle is a derived class of Shape */
}
Circle cobj = new Circle();          /* create an object */
cobj.setColor(Color.RED);
cobj.setLineStyle(LineStyle.Solid);
```

Coding Convention:
class name: capital letter, noun
action: starting with lower case, verb

Inheritance
reuse attributes from base class

```
class Shape {
   private Color s_Color;
   private LineStyle s_LineStyle;
   private int s_thickness;
}
class Circle extends Shape {
}
Circle cobj = new Circle();
cobj.s_Color   ⇐ the color attribute
```

Coding Convention: attribute: start with a lower case letter of the class' name, followed by an underscore

Polymorphism

Objects of Different Derived Classes Behave Differently

```
class Shape {
   public float getArea();
}
class Circle extends Shape {
   private float c_radius;              // attribute unique to this derived class
   public float getArea() { return (c_radius * c_radius * Math.PI); }
     /* the formula to calculate area is unique for each derived class */
}
class Rectangle extends Shape {
   private float r_width, r_height;
   public float getArea() { return (r_width * r_height); }
}
```

ECE 462
Object-Oriented Programming using C++ and Java

Lecture 2

Yung-Hsiang Lu

yunglu@purdue.edu

Polymorphism

Objects of Different Derived Classes Behave Differently

```
class Shape {
   public float getArea();
}
class Circle extends Shape {
   private float c_radius;              // attribute unique to this derived class
   public float getArea() { return (c_radius * c_radius * Math.PI); }
     /* the formula to calculate area is unique for each derived class */
}
class Rectangle extends Shape {
   private float r_width, r_height;
   public float getArea() { return (r_width * r_height); }
}
```

Reuse or Not?

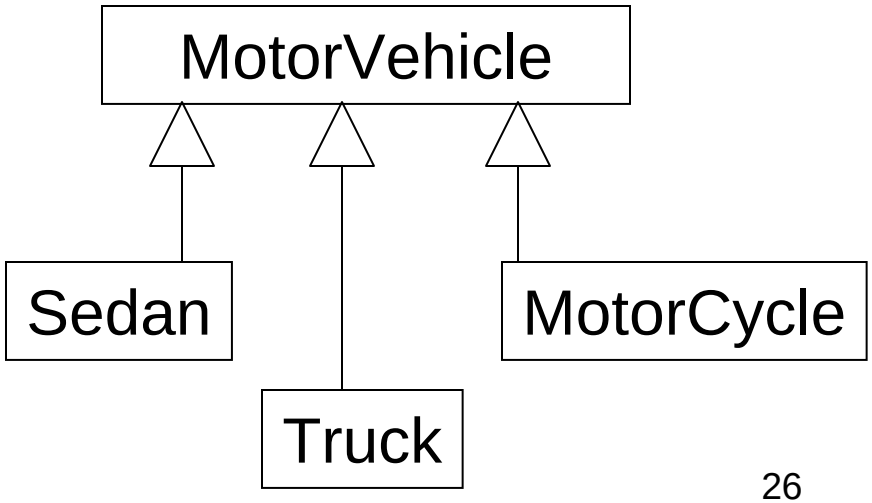
- Attribute:
  - If an attribute is shared by all derived classes, the attribute should be declared in the base class. example: color, line style, thickness.
  - If an attribute is unique to a class, it should be declared in this derived class, example: radius for circle.
- Behavior (member function, method, message):
  - If a method is available (interface) to all derived classes, such as getArea and setLineStyle, it should be declared in the base class.
  - If the method's implementation is applicable to all derived classes, it should be implemented in the base class.
  - If the implementation is unique to each derived class, it should be implemented in the derived classes.

Base or Derived Classes

- base: more general, including only common attributes and methods, “smaller” (do not call it superclass)
- derived: more specific, including additional attributes and methods, “larger” (do not call it subclass).
- **Any object of a derived class is also an object of the base class, false in the other direction.**
- Human (base): Student (derived) $\Rightarrow$ a student object includes additional attributes, such as student ID, school name, number of courses taking ... and additional methods, such as submitHomework ... A student object is also a Human object.

More Examples about Base / Derived

- attributes:
  - base, MotorVehicle: engine size (int) , brand (string)
  - derived, Sedan: sunroof or not (boolean)
  - derived Truck: towing capacity (int)
  - derived MotorCycle ...
- methods:
  - MotorVehicle: accelerate
  - Truck: load cargo
  - ...



Self Test

- In the following pairs, which one should be the base class and which one should be the derived class?

- Computer / DesktopComputer
- CollegeStudent / Student
- Teacher / CollegeProfessor
- DrinkingWater / Liquid
- Metal / Iron
- Eagle / Bird
- JetPlane / Airplane
- Furniture / Chair
- Boat / SteamBoat
- Electronics / Computer / LaptopComputer

- Computer (B) / DesktopComputer
- CollegeStudent / Student (B)
- Teacher (B) / CollegeProfessor
- DrinkingWater / Liquid (B)
- Metal (B) / Iron
- Eagle / Bird (B)
- JetPlane / Airplane (B)
- Furniture (B) / Chair
- Boat (B) / SteamBoat
- Electronics (B) / Computer (D1) / LaptopComputer (D2)
  $\Rightarrow$ 3 layers of class relationship

- The relationship among base and derived classes is called the **class hierarchy**.
- Class hierarchy is a **programming concept**, **not** biological classification. Design a class hierarchy based on the need of the program. For example, whether "canFly" is a property of Bird and whether Chicken should be a derived class of Bird.

Self Test

- In the following base / derived classes, mark which method should be declared / implemented in base and which should be declared / implemented in derived.

- Computer / Laptop: printDocument / chargeBattery
- MotorVehicle / Truck: accelerate / decelerate
- Building / OfficeBuilding: turnOnHeater / shutOffWater
- Human / Student: askAge / askGender
- Telephone / MobilePhone: dialNumber / computerRoamRate
- Construction / Bridge: getSpanLength / getHeight

- Computer / Laptop: printDocument (B) / chargeBattery (D)
- MotorVehicle / Truck: accelerate (B) / decelerate (B)
- Building / OfficeBuilding: turnOnHeater (B) / shutOffWater (B)
- Human / Student: askAge (B) / askGender (B)
- Telephone / MobilePhone: dialNumber (B) / computerRoamRate (D)
- Construction / Bridge: getSpanLength (B) / getHeight (B)

- These examples illustrate that in many cases, the methods can be declared in the base classes and **reused** in the derived classes.
- As shown in the example of getArea, the implementation may be unique in derived classes.

Encapsulation
Hiding Information Inside Objects

- Objects can interact through only **public** methods. Methods and attributes can be declared as private or protected.
  - private: accessible (read / write attributes, or call methods) by only the class
  - protected: accessible by the class or its derived classes
- Objects do not know other objects' internal structures, for example, how can a Shape object calculates its area.
- Encapsulation ensures that objects' attributes cannot be accidentally modified externally and allows future improvement without affecting the interfaces.
- If an object wants another object to do something, the first object **sends a message** to the second object.

Four Key Concepts about OOP
(beginning of Ch 3)

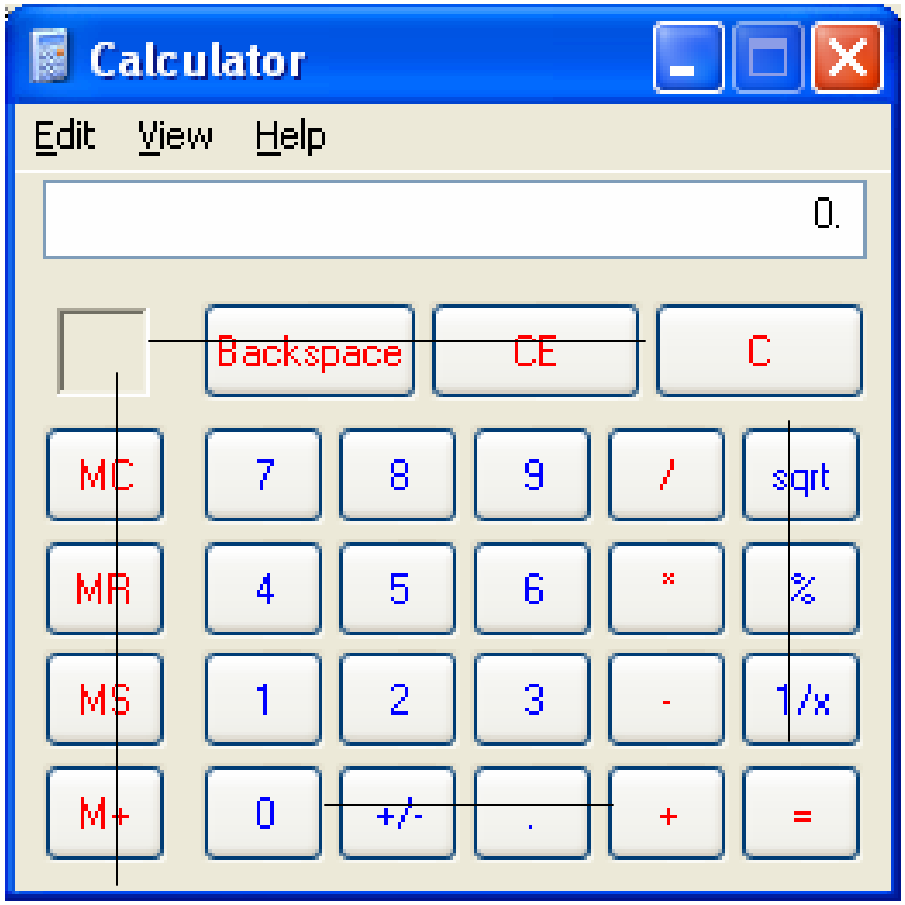
- class
- encapsulation
- inheritance
- polymorphism

Demonstration of Building Graphical User Interface Using Netbeans

in ECN Linux machine, netbeans is available at /usr/opt/bin/netbeans

Lab 1: Java GUI using netbeans

- create a calculator that can handle +, -, *, and /.
- allow multiple digits
- use integer in Java
- do not worry about
  - fractions
  - overflow or underflow
  - divided by zero
  - set or reset memory
  - backspace
  - sign toggle
  - square root



What is Method / Message

A **message** is a request **to an object** to do something. An object can receive a message if it is declared as a method in a base class or the corresponding class.

```
class MotorVehicle {
    public void accelerate(int val);
}
class Truck extends MotorVehicle {
    public void towTrailer(Trailer tra);
}
Truck t = new Truck();
t.accelerate(50);                    t.towTrailer(...);
```

Interactions Among Objects

- An OO (object-oriented) program, many objects are created and they interact by “sending messages”, for example,
  - A Driver object sends a message “accelerate” to a MotorVehicle object.
  - An Instructor object sends a message “submitHomework” to a Student object.
  - A Caller object sends a message “callNumber” to a MobilePhone object.

```
class ClassName {
   public void doSomething(...)
}
ClassName anobject;
anobject.doSomething(...);      /* this object is asked to do something */
```

```
class Shape {
   abstract public float getArea();  // interface, no implementation
}
class Circle extends Shape {
   private float c_radius;              // attribute unique to this derived class
   public float getArea() { return (c_radius * c_radius * Math.PI); }
     /* the formula to calculate area is unique for each derived class */
}
class Rectangle extends Shape {
   private float r_width, r_height;
   public float getArea() { return (r_width * r_height); }
}
```

An abstract method is declared as an **interface** and it must be **implemented** in the derived classes.

Message

- **not** a network concept
- the mechanism to interact with an object
  - ask a bridge about its length
  - turn on a light (no parameter)
  - accelerate a car (parameter?)
  - add a customer to a database (parameter = customer)
  - ask a customer of the credit number
- disallowed messages are checked at compile time: compiler error if you ask a light bulb to accelerate.

Self Test

Determine which object sends messages:
obj.message(arg) means a message is sent to object obj with parameter arg.

- A Driver object (dobj) sends an “accelerate” message to a Car object (cobj)
- A Teacher object (tobj) sends a message to a Student object (sobj) to submit homework.
- A Computer object (cobj) sends a Packet object (pobj) to a Network object (nobj).

- A Driver object (dobj) sends an “accelerate” message to a Car object (cobj)

  $\Rightarrow$ cobj.accelerate();
- A Teacher object (tobj) sends a message to a Student object (sobj) to submit a Homework object (hobj).

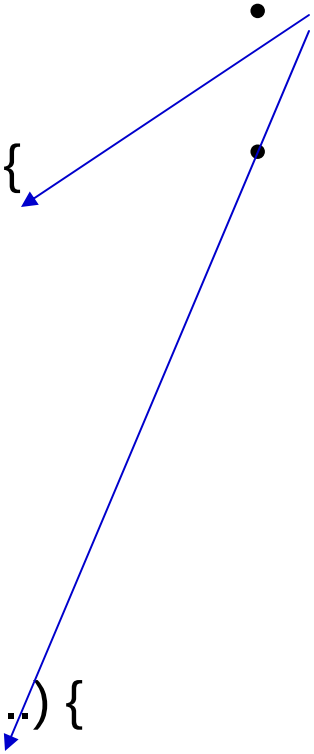
  $\Rightarrow$ sobj.submit(hobj);
- A Computer object (cobj) sends a Packet object (pobj) to a Network object (nobj).

  $\Rightarrow$ nobj.send(pobj);

The object is the **recipient** of the message. Where is the sender? It is implicit by the location of the message.

```
class Driver {
    private Car d_car;
    public void driveCar(...) {
        d_car.accelerate();
    }
}

class Teacher {
    private Student t_stu;
    public void teachClass(...) {
        t_stu.submit(hobj);
    }
}
```



- The “sender” is the method where a message is sent.
- The recipient must be initialized (not discussed yet) before sending any message.

Object Type

- If an object is declared to be an **instance** of a class, the object can refer to an object of a derived class.
- example

```
Human obj1;
Student obj2;
obj1 = obj2; // assign a student to obj1, ok
obj2 = obj1; // wrong, a human may not be a student
```

aobj = bobj $\Rightarrow$ assign bobj to aobj, aobj (e.g. Human or Student) can be bobj (e.g. Student), aobj is in the same class or more general (base class).

```
class Shape {
   abstract public float getArea();  // interface, no implementation
}
class Circle extends Shape {
   private float c_radius;              // attribute unique to this derived class
   public float getArea() { return (c_radius * c_radius * Math.PI); }
}
class Rectangle extends Shape {
   private float r_width, r_height;
   public float getArea() { return (r_width * r_height); }
}
```

Shape sobj = new Rectangle;

sobj.getArea() ⇒ call Rectange's getArea

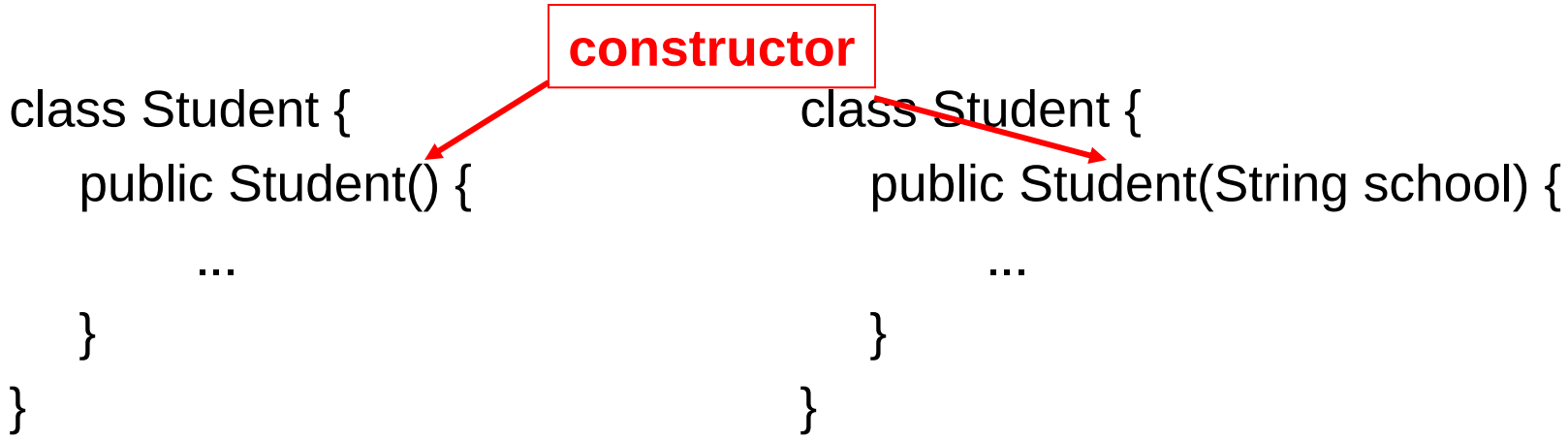
sobj = new Circle;

sobj.getArea() ⇒ call Circle's getArea

Rectangle robj = new Shape;
// WRONG
Circle cobj = new Rectangle;
// WRONG

Object Creation

- Programmer-provided “constructor”: a method with the **same name as the class**. The method may take input arguments.
- Constructors can create objects in consistent ways and no attributes are left uninitialized.
- In C++ and Java, a function can have different numbers and types of input parameters (called **overloading**).



```
// ---------------------- User.java
class User {
    private String name;
    private int age;
    public User( String str, int yy ) { name = str;  age = yy; }
    public void print() {
        System.out.println( "name: " + name + "  age: " + age );
  }
}
class Test {
    public static void main( String[] args ) {
        User u = new User("Zaphod", 23 );
        u.print();
    }
}
```

similar to (int argc, char * argv[]) but Java arrays know their lengths