# 05 Type Systems

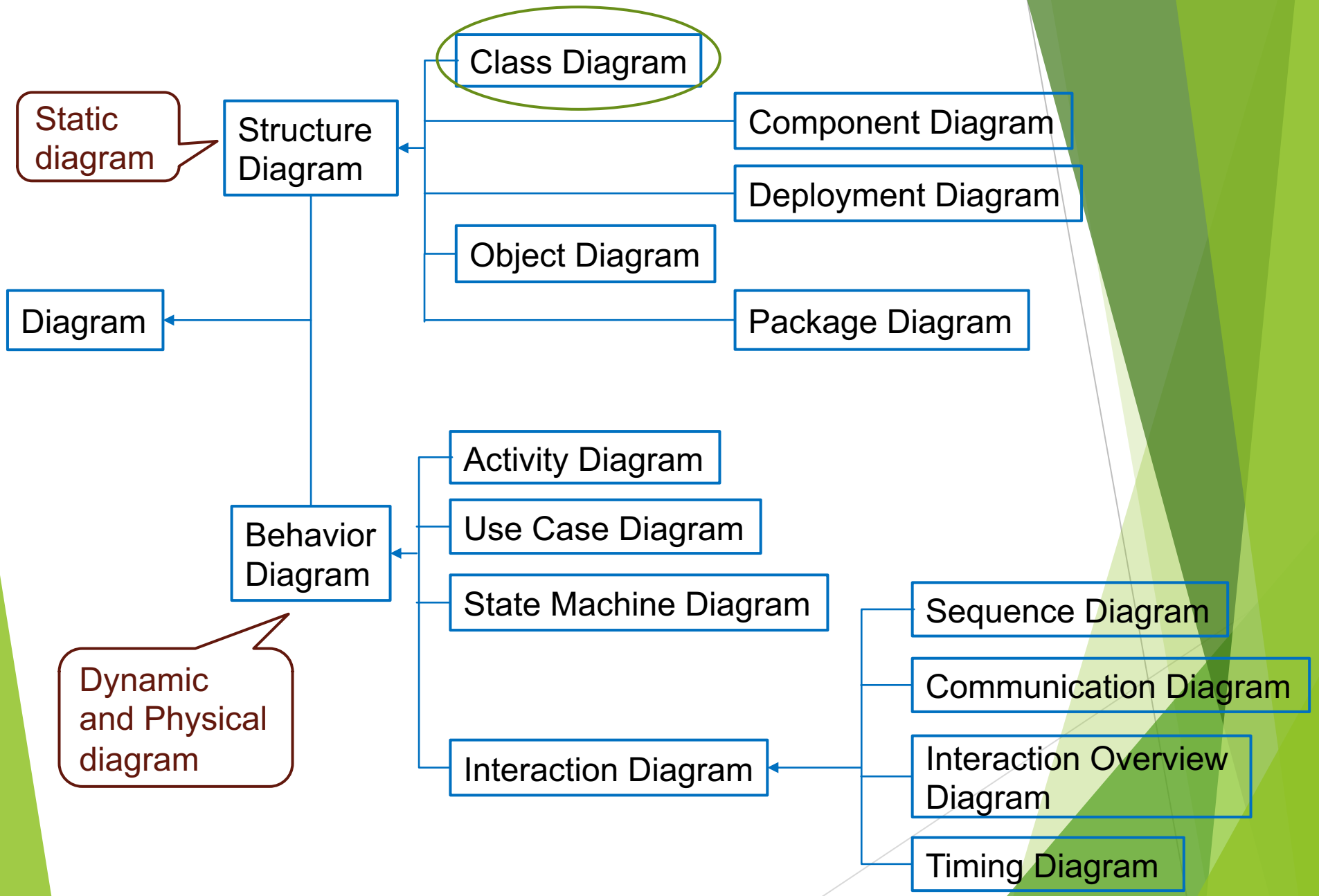
*Introduction to OOA OOD and UML*

*2022 Spring*

College of Information Science and Engineering

Ritsumeikan University

Yu YAN


Class Diagram
Static diagram
Structure Diagram
Component Diagram
Deployment Diagram
Object Diagram
Diagram
Package Diagram
Activity Diagram
Behavior Diagram
Use Case Diagram
State Machine Diagram
Dynamic and Physical diagram
Sequence Diagram
Communication Diagram
Interaction Diagram
Interaction Overview Diagram
Timing Diagram

# Outline

- The Definition of Type Systems
- Static and Dynamic Type Systems
- Polymorphism
- Type Casting
- Summary and Class Vacabularies
- Exercise 05

# What Is a Type System?

- A type system is a set of rules that assign a **property** (called type) to various constructs in a computer program, which consist of *variables*, *expressions*, *functions* and *modules*.
- The main purpose of a type system is to reduce possibilities for bugs in computer programs by defining interfaces among different parts of a computer program and checking that parts have been connected in a consistent way.
- An example of a type system in use is declaring that a variable will always hold a value of a particular type

```
int i;
Car bmw;
i = 2; // assign a value
double d = 1.0; // declare and assign
```

# Advantages of Type Systems

- Type system rules avoid us misusing values (primitives and objects).
  - This is done by forcing us to declare how we intend to use a value before we actually use it – this allows compilers and run-time systems to spot potential abuses before they happen
- A type system can ensure that we provide some documentation of the code
- A type system can also improve run-time performance
  - The compiler and run-time system have more information about what the code intends to do (compiler) or what it actually is doing (run-time system)

# Outline

- The Definition of Type Systems
- Static and Dynamic Type Systems
- Polymorphism
- Type Casting
- Summary and Class Vacabularies
- Exercise 05

# Static Type Systems

- A programming language is said to use "static typing" when type checking (type system) is performed during compile-time (done by the compiler)
  - In static typing, types are associated with variables, not values
  - Static typing allows many type errors to be caught early in the development cycle
  - Static typing eliminates the need to repeat type checks every time the program is executed
  - Compiled code is more efficient , compact and faster
- Languages: C, C++, C#, Java, …

# Static Typing – Examples

```
float f;
double d;
d = 1.0; // assign a value
f = 1.0; // possible
```

And

```
float f;
double d;
d = 1.0; // assign a value
f = 1.0f; // preferable
```

```
public void addEmployee(Employee anEmployee){
    ...
    pay = anEmployee.getPayrollNumber();
    ...
}
```

The type of the parameter is "Employee" and the type of returned value is "void"

```
aPayroll.addEmployee(new Banana());
```

"Wrong parameter usage" was reported by compiler, resulting a compiler error

# Dynamic Type Systems

- A programming language is said to use "dynamically typed (dynamic)" when the majority of its type checking (type system) is performed at run-time (done by the run-time system)
  - In dynamic typing, types are associated with values, not variables
  - There is no needs to declare type in advance
  - Dynamic typing is more flexible: type is generated based on run-time data
  - However, dynamic typing might have more run-time errors, and more testing needs to be done
- Languages: Python, JavaScript, …

# Comparison Between Java and Python

- In static typing, once a variable is set to a type, it can not be changed later, for example:

```
// the variable "str" statically typed as a "string"
String str = "Hello";
// an error would be thrown since "str" is supposed to be a "string" only
str = 5;
```

- In dynamic typing, after a variable is set to a type, it can be changed later, for example:

```
# the variable "str" is linked to a "string" value
str = "Hello";
# now "str" is linked to an integer value
str = 5;
```

# Outline

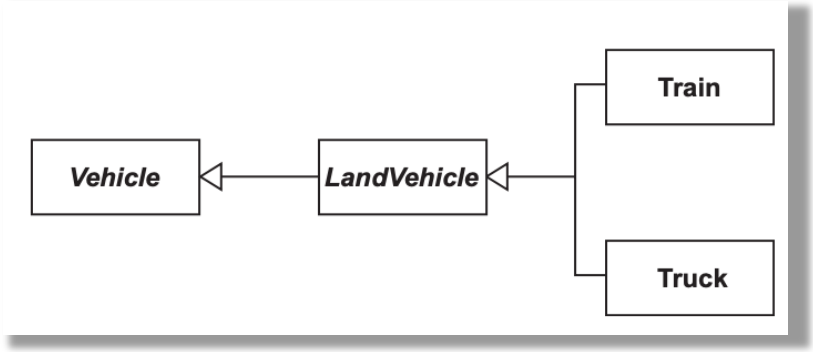
- The Definition of Type Systems
- Static and Dynamic Type Systems
- Polymorphism
- Type Casting
- Summary and Class Vacabularies
- Exercise 05

# Polymorphic Variables (1)

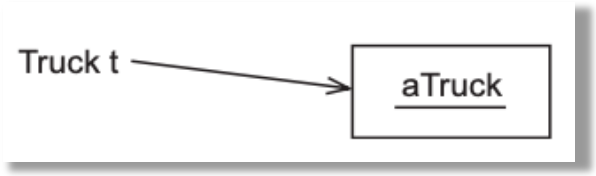
- Polymorphic means "having many shapes":
  - A polymorphic variable refers to different types of value at different times
  - A polymorphic message has more than one method associated with it.
- The polymorphism of variables is controlled by inheritance
  - Polymorphism allows us to attach a variable to a subclass object. But we can't go the other way round.

# Polymorphic Variables (2)

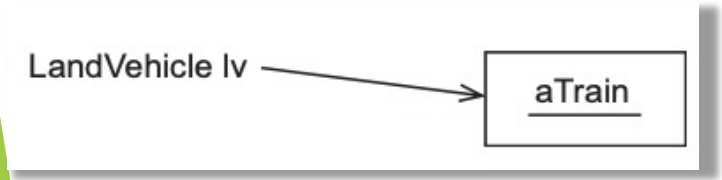
- Assignment (1)




Truck inheritance



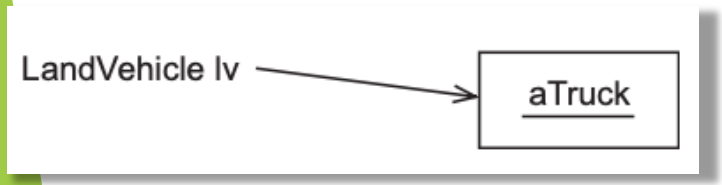

Truck t;
t = new Truck();




LandVehicle lv;
lv = new Train();

1. Class "Orange ()" is not in our inheritance.
2. Class "Vehicle ()" is upper level of our inheritance.

lv = new Orange();



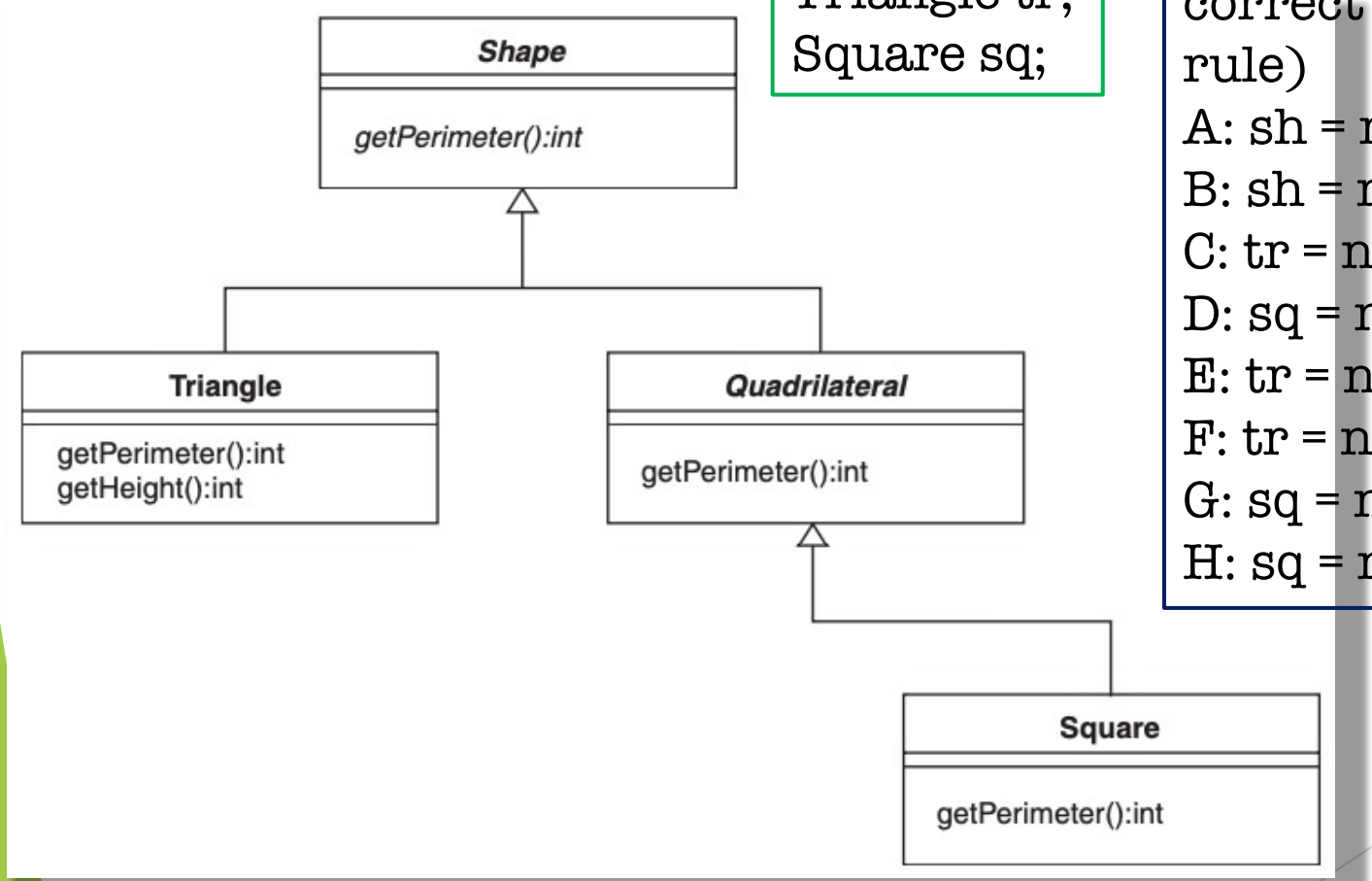

lv = new Truck();

lv = new Vehicle();

"lv" is a polymorphic variable/reference

# Polymorphic Variables (3)

- Assignment (2)




```
Shape sh;
Triangle tr;
Square sq;
```

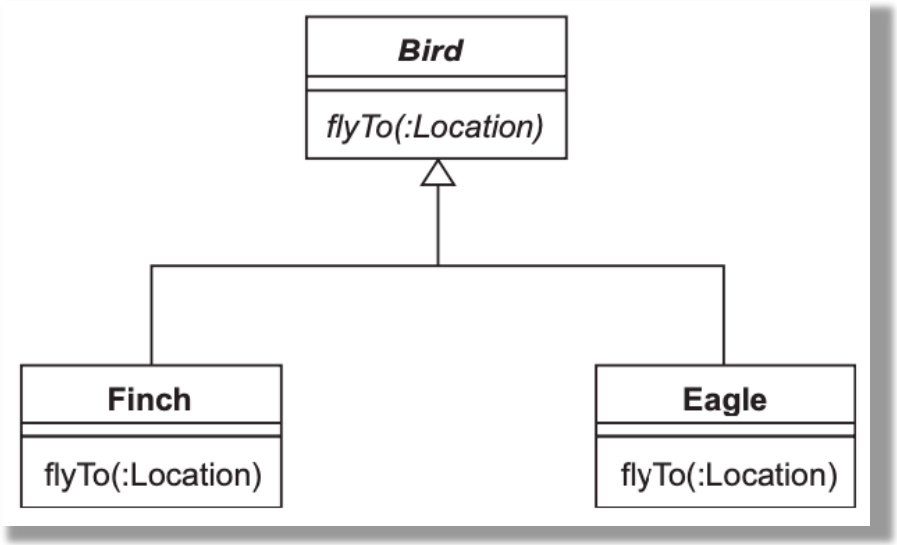
Which assignments are correct ? (use "kind of" rule)

A: sh = new Triangle();
B: sh = new Square();
C: tr = new shape();
D: sq = new shape();
E: tr = new Triangle();
F: tr = new Square();
G: sq = new Square();
H: sq = new Triangle();

# Polymorphic Messages (1)

- A polymorphic message has more than one method associated with it
- Any message, in a pure OO language, can have more than one method associated with it. This happens:
  - Either because the methods appear independently on more than one class
  - Or because a method is redefined by subclasses
- The polymorphism of messages is controlled by inheritance too

# Polymorphic Messages (2)




Polymorphic animal messages

```
Bird b;
b = new Finch();
b.flyTo (someLocation);
b = new Eagle();
b.flyTo(someLocation);
```

There is only one bird. "b" is a reference pointing to the bird. "flyTo" is a polymorphic message

```
Cat tiddles, tom;
tiddles = new Cat("Hfrrr");
tom = tiddles;
```

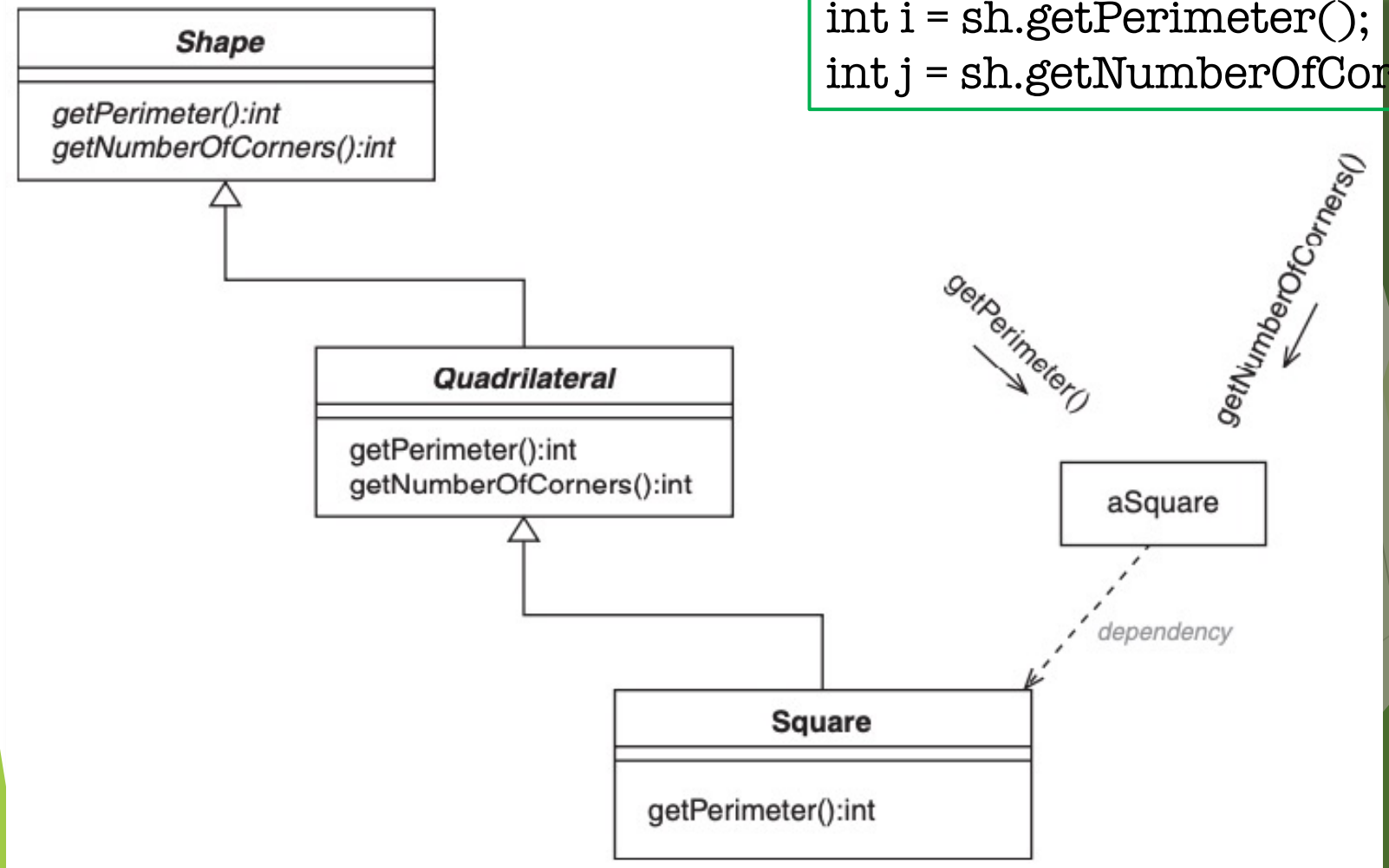
There is only one cat. "tiddles" and "tom" are two references both pointing to the cat.

# Dynamic Binding (1)

- Dynamic binding means attaching a message to method at run-time
  - Dynamic binding is the way that OO languages cope with polymorphic variables and redefined methods
  - Dynamic binding is related with polymorphism and inheritance
- Other definitions and related terms
  - Dynamic binding is also known as dynamic dispatch, late binding or run-time binding
  - Dynamic binding is the process of linking procedure call to a specific sequence of code (method) at run-time. It means that the code to be executed for a specific procedure call is not known until run-time

# Dynamic Binding (2)

```
Shape sh = new Square();
int i = sh.getPerimeter();
int j = sh.getNumberOfCorners();
```



# Polymorphism Guideline

- Always program using as high a level of abstraction as you can
  - Always declare the type of your fields, local variables and method parameters to be the highest class possible in the inheritance hierarchy, then let polymorphism do the rest
- These rules are useful for code re-usability

# Outline

- The Definition of Type Systems
- Static and Dynamic Type Systems
- Polymorphism
- Type Casting
- Summary and Class Vacabularies
- Exercise 05

- Converting a value from one type to another is called casting
  - In a statically typed language, when we pass a value from one context to another, we need to be sure that the new context is compatible with the old one
- There are three situations in which a value changes context:
  - Expression evaluation

If the casting is done without special programming and ambiguity, it is call implicit casting

2+4

integer adds integer: no casting needs

3.14+2

real number adds integer: it will produce a real number. Compiler will convert the integer into a real number automatically

"The date is" + aData

String adds object: it will produce a string. Compiler will translate the "aData" into a String by sending it the toString message.

- Assignment

```
Person p = new Person()
```

no casting needs

```
Person p = new Employee()
```

Person pointers are not represented differently from Employee pointers. But in order to make sure the conversion is safe: Person needs to be a direct or indirect superclass of Employee

- Parameter pass

```
aUniversity.enrollPerson (new Employee())
```

The parameter is declared to be Person

```
aLiquid.setBoilingPoint(100)
```

The parameter is declared to be float

# Explicit Casting (1)

- An implicit cast is possible if the new context is **"wider"** than the old context
  - **"wider"** means the new context can accommodate all possible values of the old context
  - For object values, **"wider"** means the new context is a direct or indirect subclass of the old context
- If the casting is done with special programming and ambiguity, it is call explicit casting
  - Explicit casting is also known as dynamic casting or down casting
  - Explicit cast allows the programmer to move from one context to a compatible, but "**narrower**", context
  - If implicit cast exists in one direction, we can force an explicit cast in the other direction
  - For object values, implicit cast depends on compiler realization

# Explicit Casting (2)

```
int i = (int) 3.75;
```

There is an implicit cast in the other direction, so it is possible to do an explicit cast in the direction: the farctional part of the real number (3.75) will be sheared off

```
Person p = new Person();
Employee e = new Employee ();
p = (Person) e;
```

# Outline

- The Definition of Type Systems
- Static and Dynamic Type Systems
- Polymorphism
- Type Casting
- Summary and Class Vacabularies
- Exercise 05

## Class Vocabularies

Type Systems, Static Typing, Dynamic Typing, Polymorphism, Casting

## Summary

- Type systems, which stop us misusing values by forcing us to declare how we intend to use a value.
  - ✓ A static type system detects abuses at compile time while a dynamic type system waits until run time
- Polymorphism, which enables a variable to hold different types of value and a message to be associated with more than one method.
- Casting between object types: with implicit casting, the compiler can automatically convert between types of variable; with explicit casting, the programmer must specify

# Outline

- The Definition of Type Systems
- Static and Dynamic Type Systems
- Polymorphism
- Type Casting
- Summary and Class Vacabularies
- Exercise 05

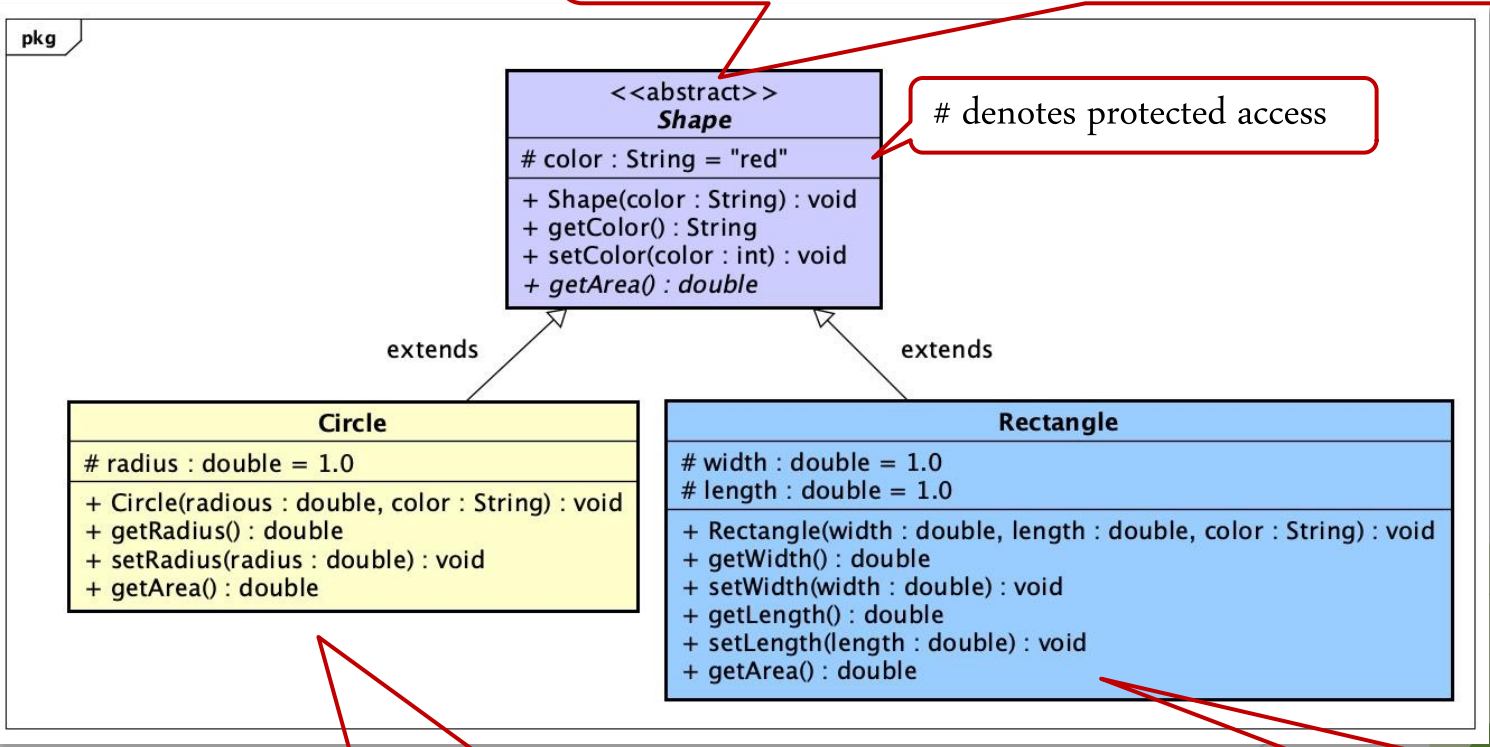
# Exercise 05

- Deadline: **2021/05/19 (Thur.) 9:00**
- Please submit your answer file to “Exercise 05” under “Assignments” tab in Manaba +R
- Please put all of the answers in one “.pdf” file. The file name will be “**UML_Ex05_Your name.pdf**”
- The maximum points for “Exercise 05” will be **5p**
- If you put a wrong file name or wrong file format, your assignment will not be evaluated. Please be careful!

# Tasks

- Task: Please draw a class diagram (with Astah UML) based on the following instructions and Example #01 (shown below), then export your diagram as a PNG/JEPG image and insert the diagram image in your report
- Example #01

"Shape" is an abstract class containing 1 abstract method: "getArea()", where its concrete subclasses ("Circle" and "Rectangle") **must provide** its implementation




"Circle" implements the abstract method "getArea()"

"Rectangle" implements the abstract method "getArea()"

# Tasks

- Requirements:
  - ✓ Please add a class “Square” in the Example #01. Think about the inheritance between the whole and the “Square” class.
  - ✓ Please also design the “Square” class including fields and methods. Be careful, each designed field and method should have its own visibility, type return value type and parameters (if the parameters are necessary).
  - ✓ Write a small documents to explain your design of the “Square” class. In the explanations, please include at least one polymorphic variable and message of the whole inheritance in the Example #01. If you don’t know how to explain a polymorphic variable and a message, please refer the slides from #12 to #16.