Computer Language

OOP 1: Class

Agenda

- OOP
- Class

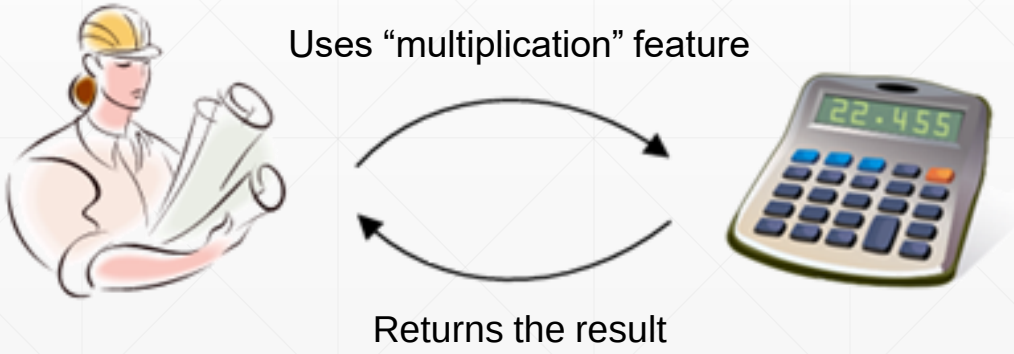
OOP

Class

OOP: Basics

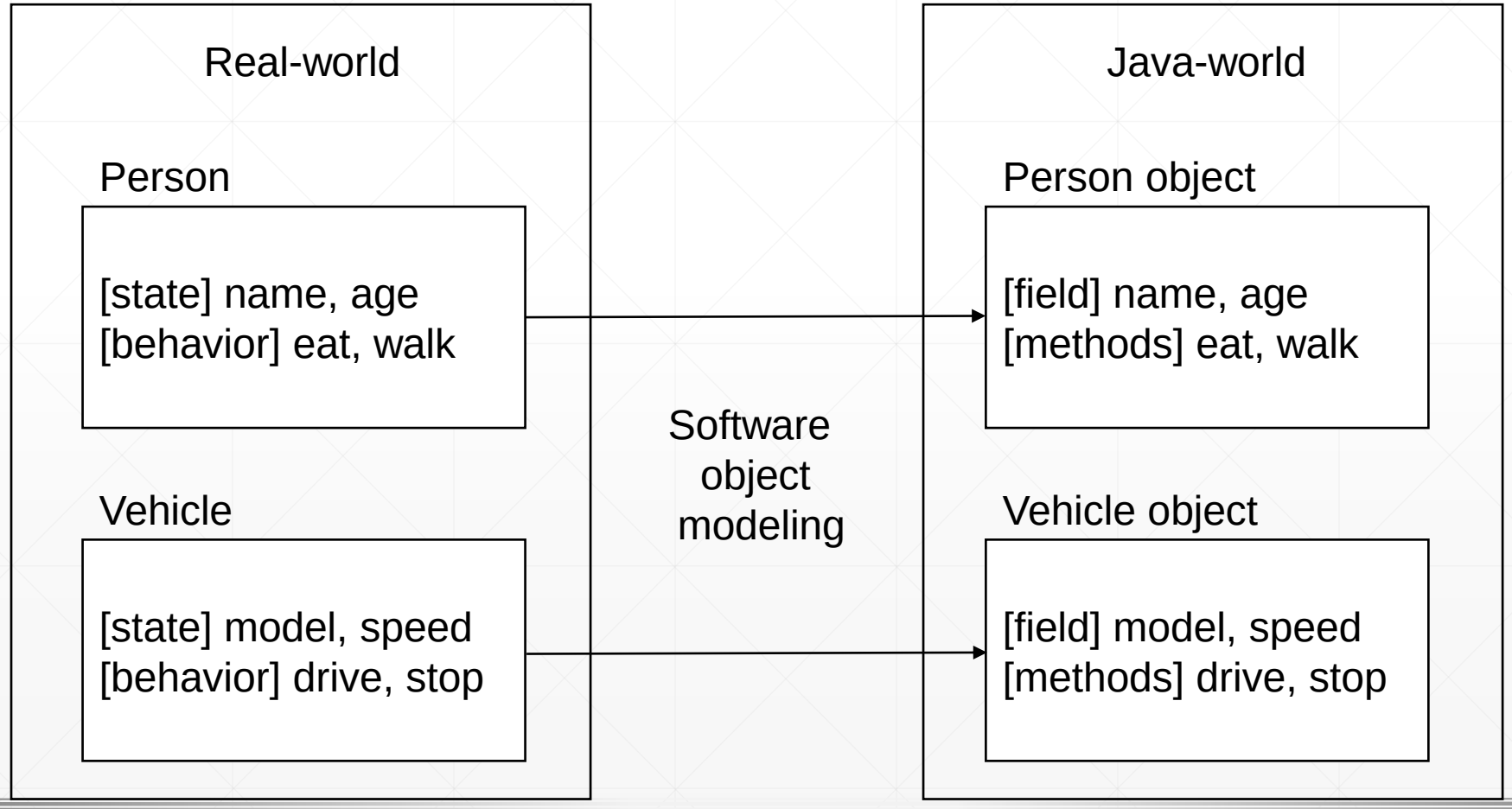
- Object-Oriented Programming (OOP)
  - The world is composed of objects (thing)

  - Characteristics of objects
    - Each object has its own state and behavior
    - Objects interact with each other



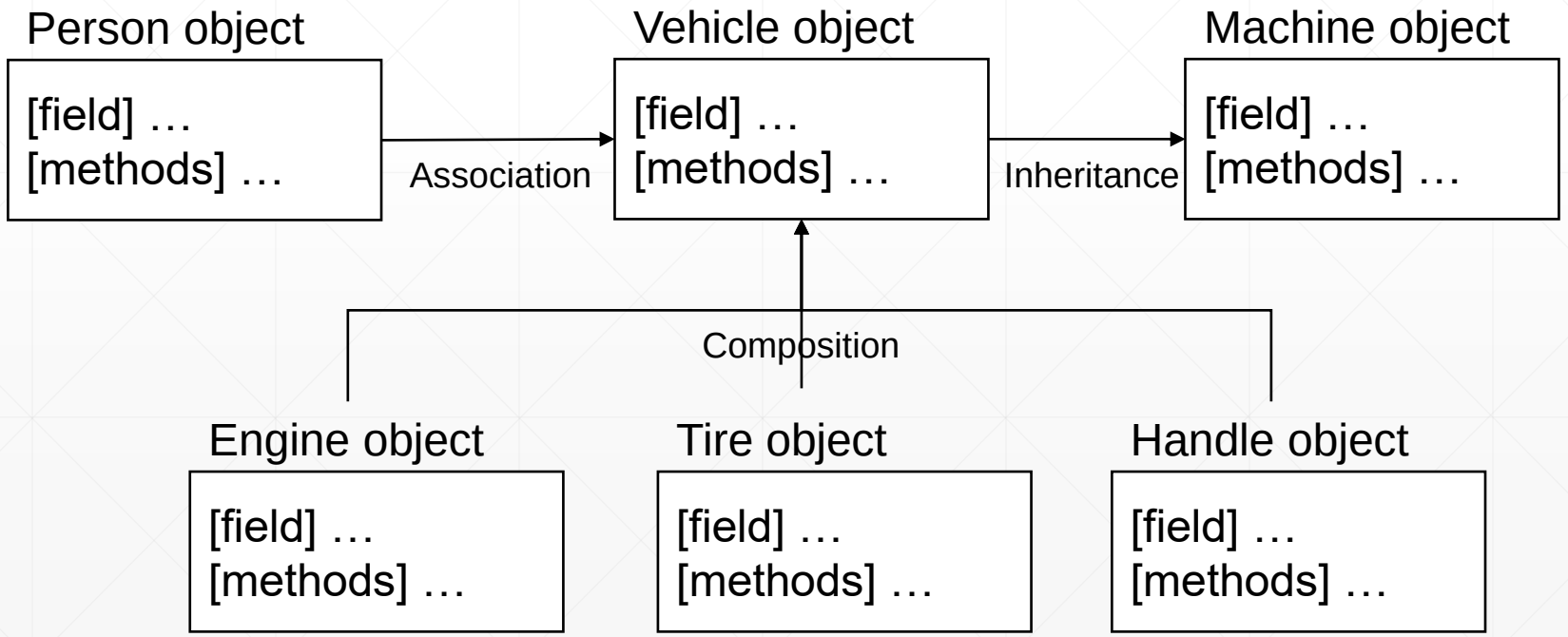
OOP: Basics (cont'd)

- Objects in Java world
  - Java object models the real-world object by defining fields (states) and methods (behaviors)



OOP: Basics (cont'd)

- Objects in Java world
  - Relationships between the objects
    - Association
    - Composition/Aggregation
    - Inheritance



OOP: Characteristics

- Encapsulation
  - Information/Implementation hiding
    - External objects cannot know the details (internal structure) of an object they want to use
    - External objects cannot access the private fields/methods of an object they want to use
    - External objects can only access the fields and methods explicitly exposed by an object they want to use
  - Access modifier is used to determine the visibility of class members (discussed later)

capsule

TV

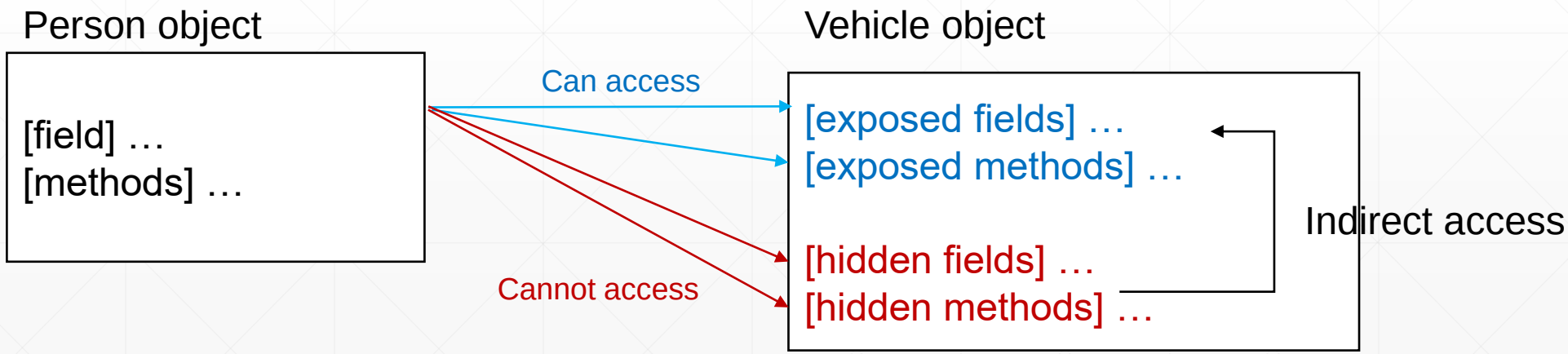
refrigerator

camera

human

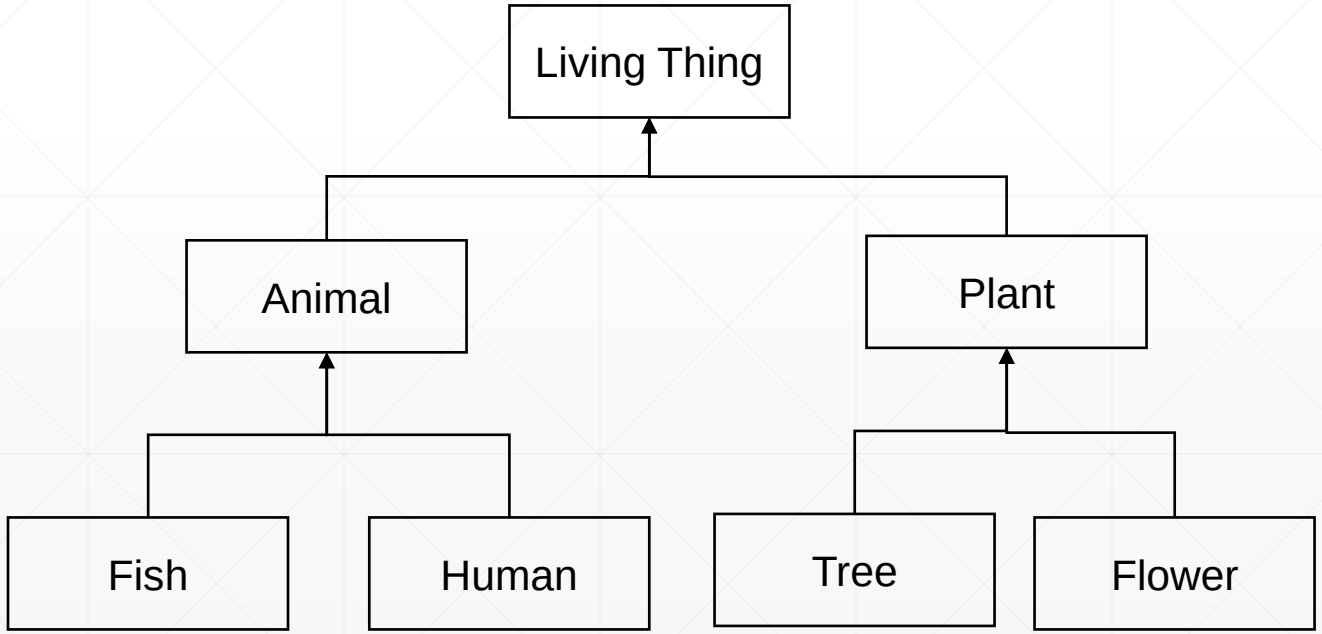
OOP: Characteristics (cont'd)

- Encapsulation: Benefits
  - Better control of class attributes and methods
  - Flexible: the programmer can change one part of the code without affecting other parts
  - Increased security of data



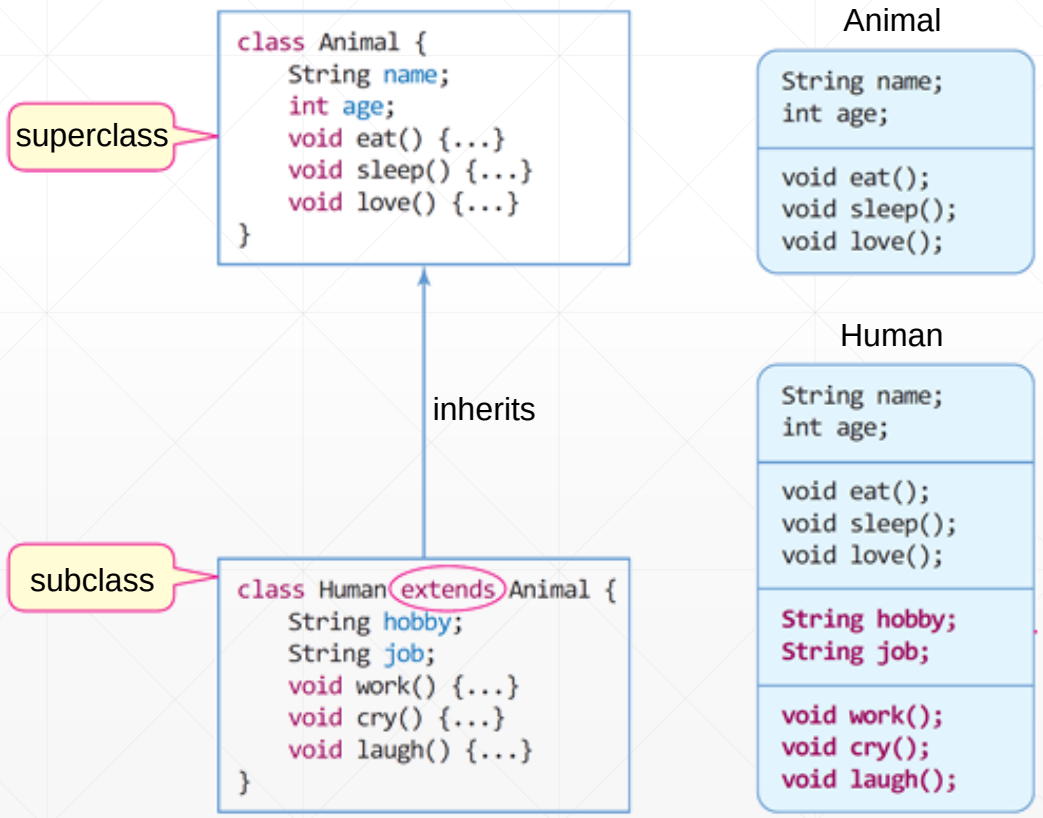
OOP: Characteristics (cont'd)

- Inheritance
  - It is possible to inherit attributes and methods from one class to another
    - Subclass: a class derived from another class (child/derived/extended class)
    - Superclass: the class from which the subclass is derived (parent/base class)



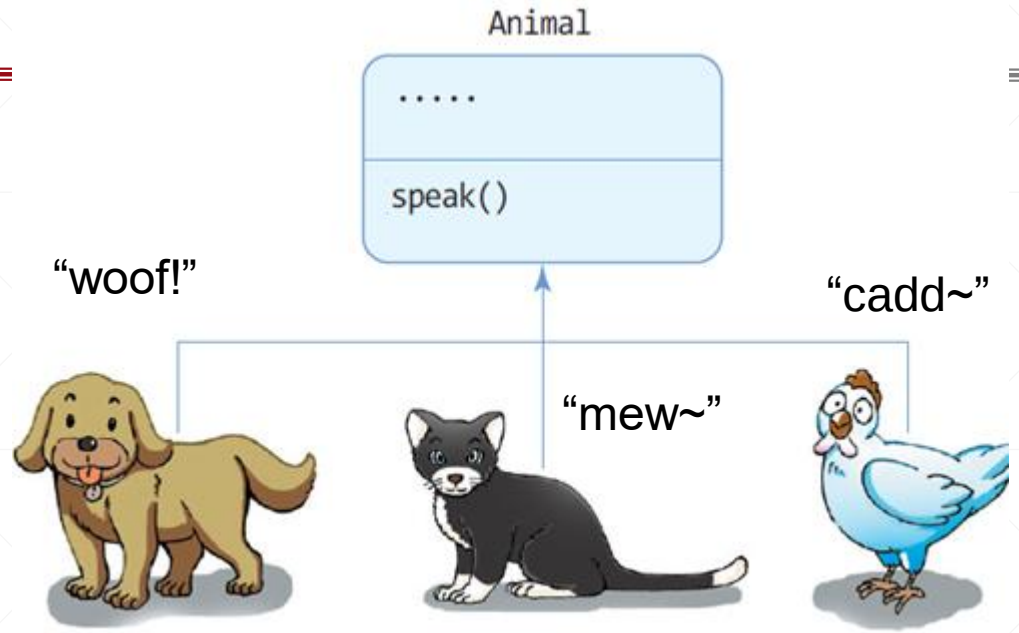
OOP: Characteristics (cont'd)

- Inheritance
  - Subclass inherits superclass's members and can extend its own features
  - We can reuse the fields and methods of the existing class without having to re-write
  - Benefits
    - Rapid implementation using existing classes
    - Reduced redundant codes
    - Better, efficient maintenance
    - Polymorphism



OOP: Characteristics (cont'd)

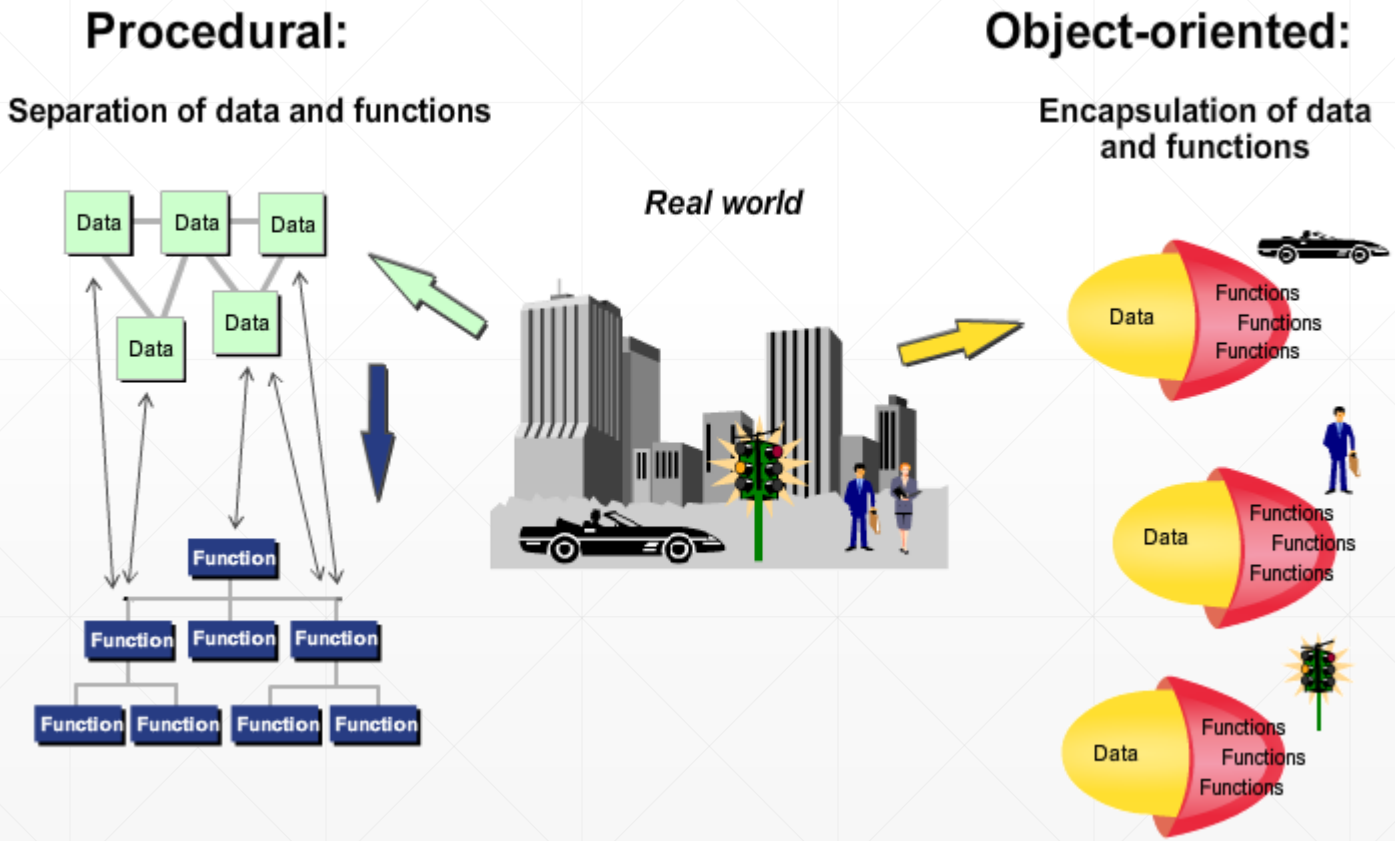
- Polymorphism
  - Definition: “having many forms“
    - “an organism or species can have many different forms or stages”
  - Subclasses of a class can **define their own unique behaviors** and yet share some of the same functionality of the parent class



- **Inheritance** lets us inherit attributes and methods from another class
- **Polymorphism** uses those methods to perform different tasks
- Inheritance and Polymorphism allow us to perform a single action in different ways
  - Benefits: flexible codes, better maintenance, etc.

OOP: Procedural vs Object-Oriented Programming

- Procedural
  - Describes a procedure of a task on the data
  - ex) C, Fortran, pascal, etc.
- Object-Oriented
  - Models a set of objects
  - Interaction between the objects
  - Ex) Java, C++/C#, Python, etc.



OOP: Procedural vs Object-Oriented Programming (cont'd)

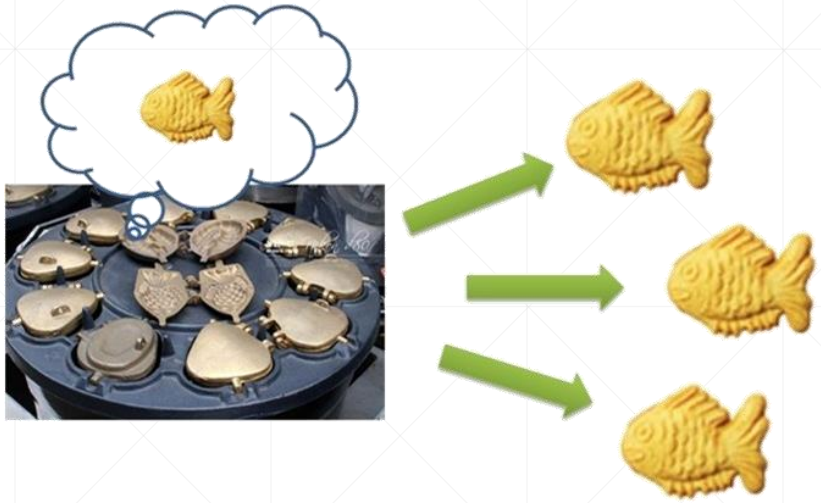
- Advantages of OOP over procedural programming
  - Modularity
    - Source code for an object can be written and maintained independently of the source code for other objects
  - Information-hiding
    - Details of internal implementation remain hidden from the outside world
  - Code re-use
    - If an object already exists, you can use that object in your program
  - Pluggability and debugging ease
    - If a particular object turns out to be problematic, you can simply remove it from your application and plug in a different object as its replacement

OOP

Class

Class

- Class
  - Frame/Blueprint to create an object
  - Includes states and behaviors of an object



- Object
  - Instance created based on the class definition
  - A concrete entity, created in program runtime, occupying a memory space
  - Multiple instances can be created from a single class
    - Class: Student Object/Instance: 201001, 201002, …

Class (cont'd)

Class: Person

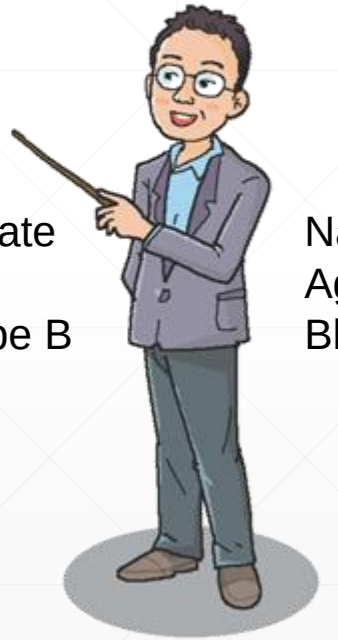
name, age, blood-type
eat, sleep, speak

Name Jane
Age 40
Bloodtype A

Object: Jane

Name Kate
Age 30
Bloodtype B

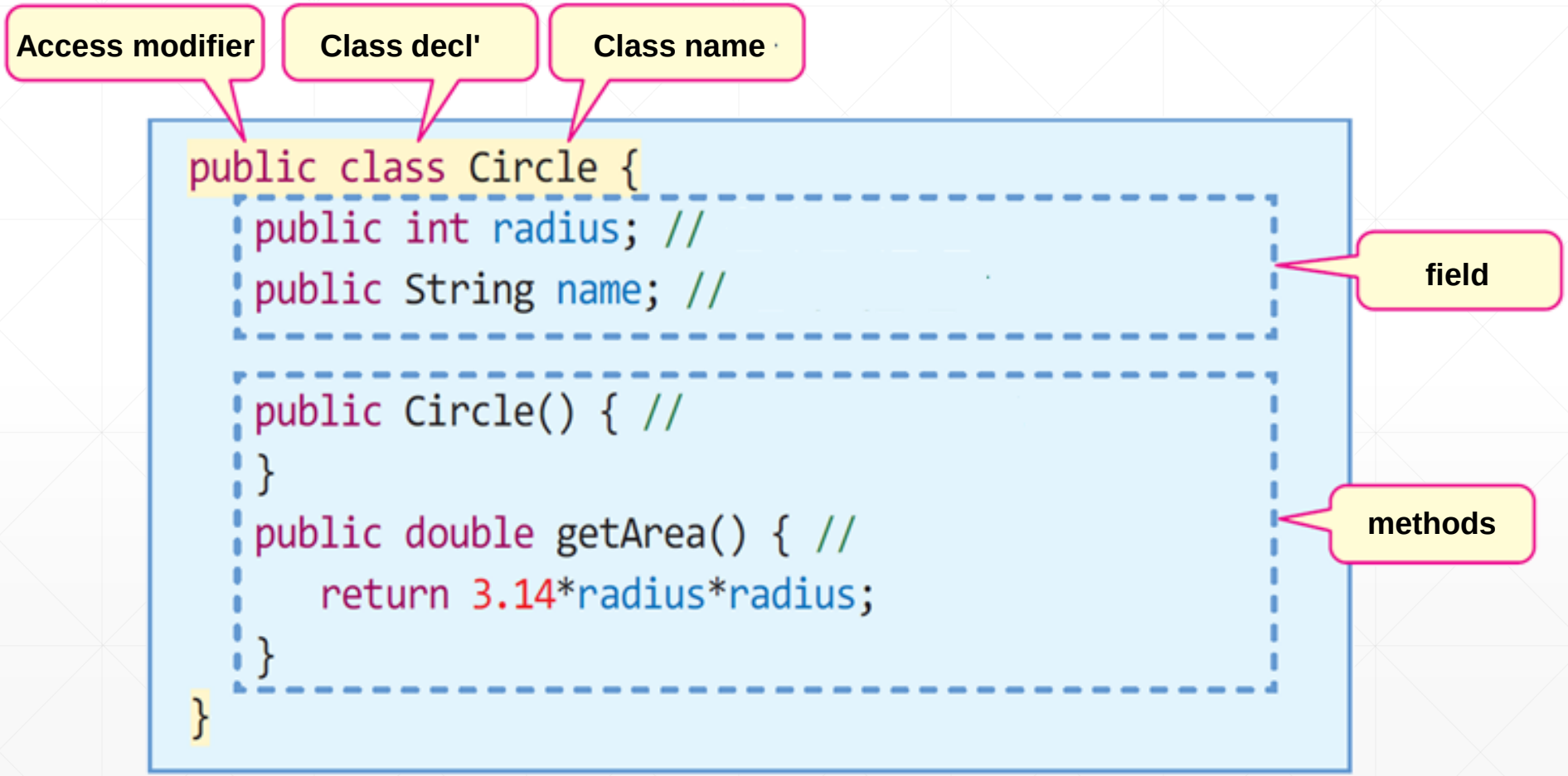
Object: Kate



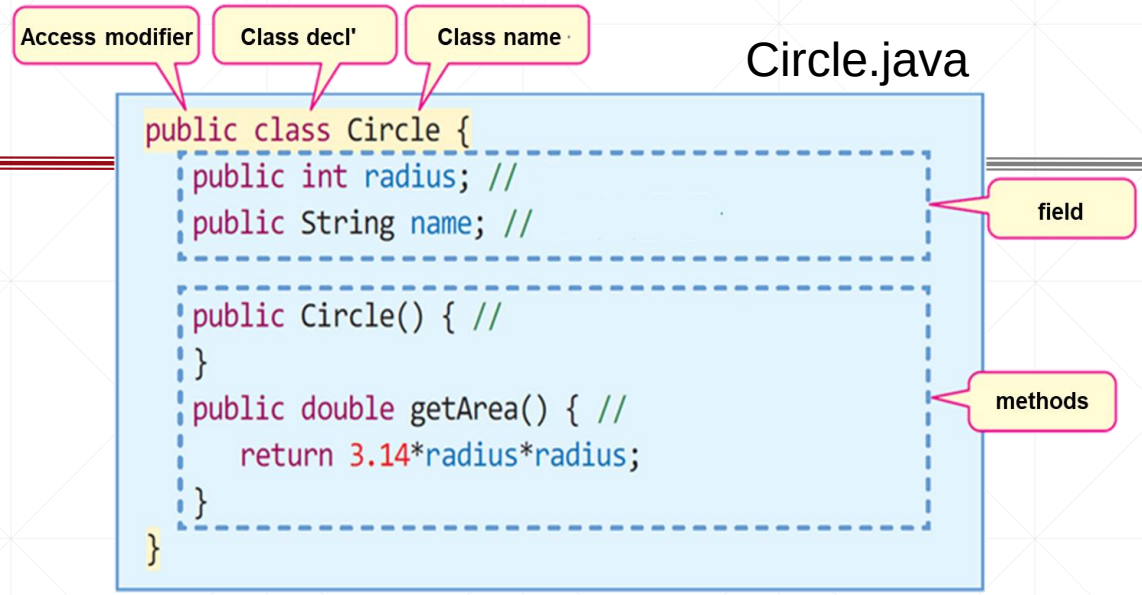
Name John
Age 35
Bloodtype AB

Object: John

Class: Structure



Class: Structure (cont'd)



- Class declaration in [ClassName].java
  - Use "class" keyword
  - Begins with "{" and ends with "}"
- Fields and methods
  - Field: member variable to store values (state)
  - Method: function in which the behavior is implemented
    - Constructor
      - Method whose name is identical to that of the class
      - Automatically invoked upon the object is created
      - Instance initialization logic can be implemented in the constructor
- Access modifier
  - Represents the accessibility of a class, fields, methods, etc.

Class: Instantiation

- Object instantiation
  - Use “new” keyword to instantiate an object

```
new className();
```

    - Constructor of an object is invoked
    - The memory address for the created object is returned
  - Sequence of object instantiation
    - Declare a reference variable for the object
    - Instantiate an object
      - Memory allocated
      - Constructor invoked
    - Access the object members

```
objReference.member;
```

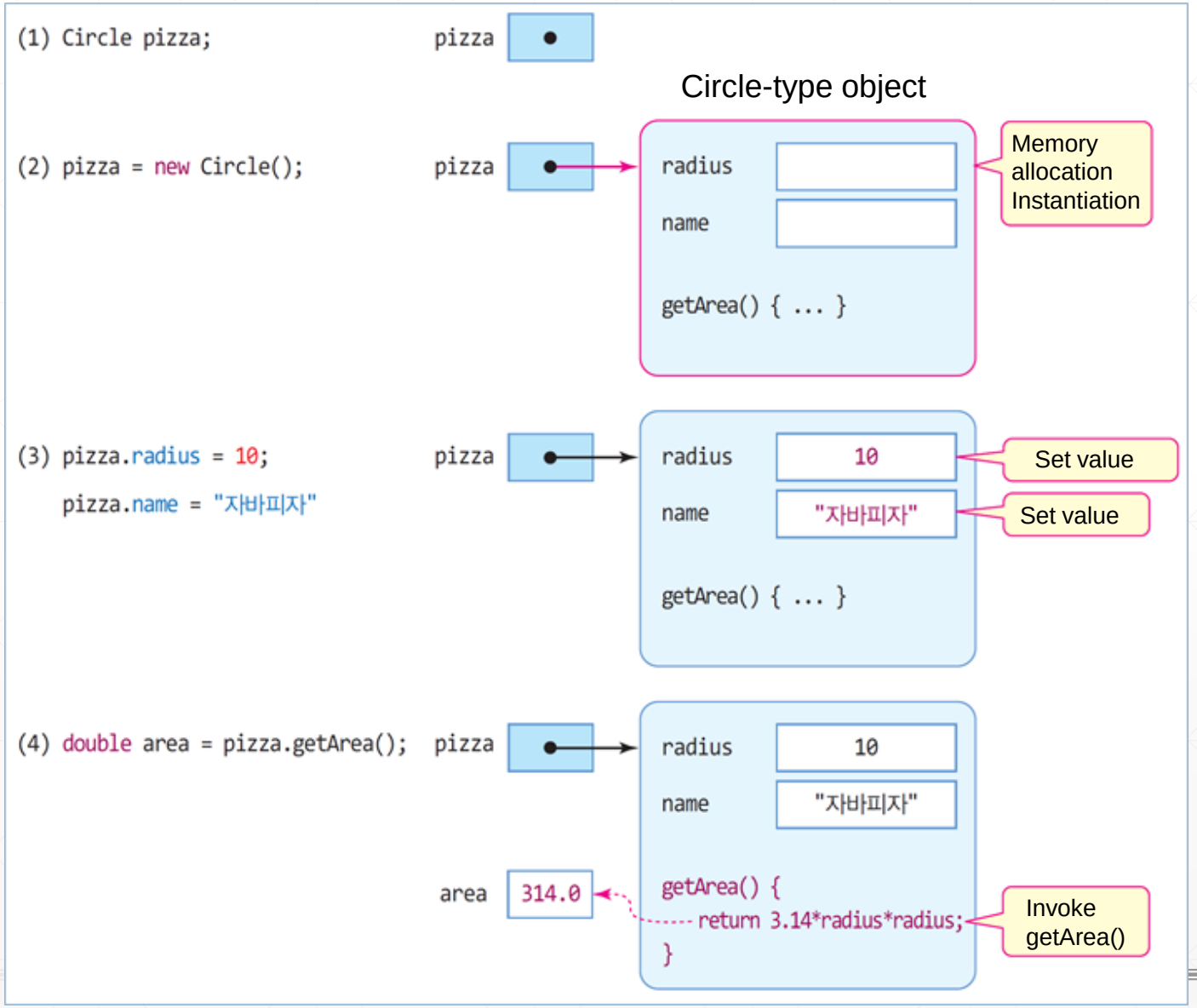
Class: Instantiation (cont'd)

1) Declaration of a reference variable

2) Object instantiation using new keyword

3) Accessing object member (field)

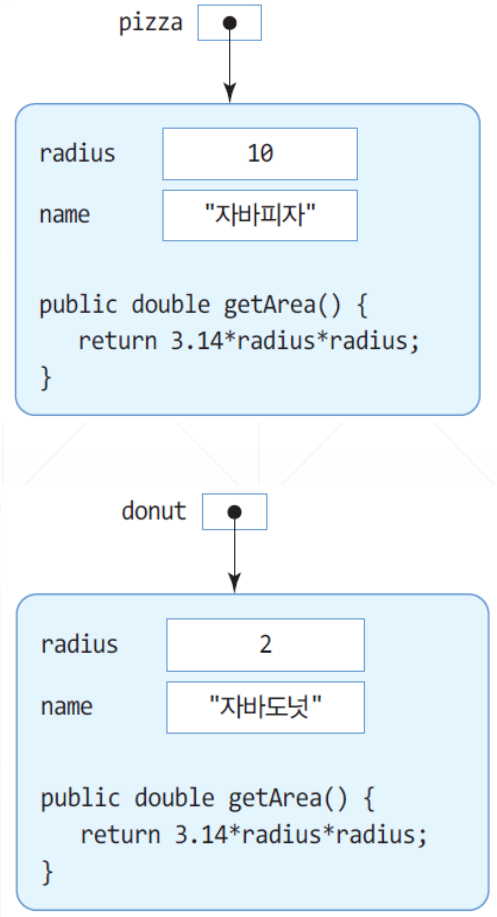
4) Accessing object member (method)



Class: Instantiation (cont'd)

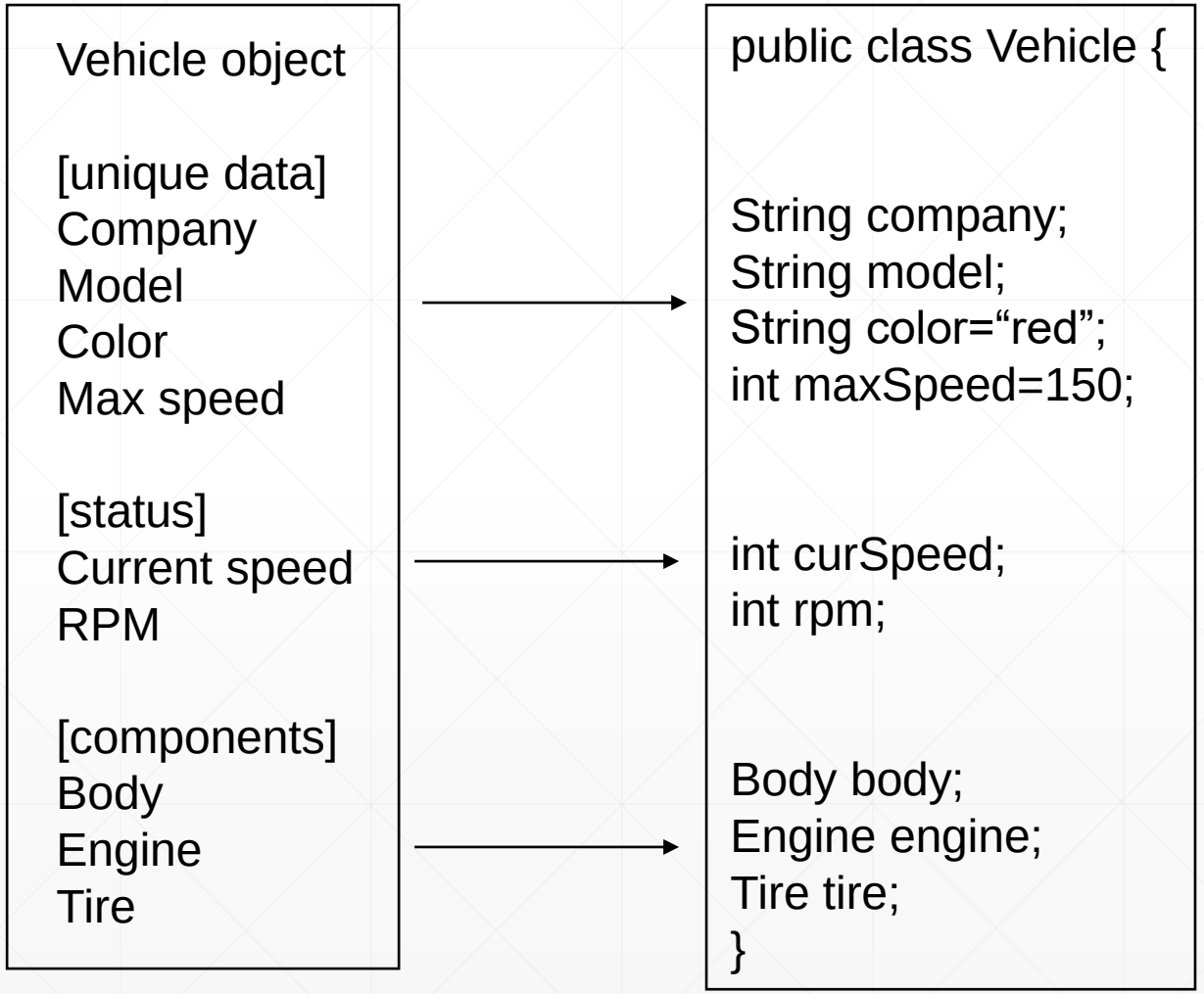
- Example)

```
public class Circle { // Circle class
   int radius;              // radius field
   String name;             // name field

   public Circle() { }      // constructor

   public double getArea() {   // method
      return 3.14*radius*radius;
   }

   public static void main(String[] args) {
      Circle pizza;
      pizza = new Circle();              // Circle object instantiation
      pizza.radius = 10;                 // set radius
      pizza.name = "자바피자";           // set name
      double area = pizza.getArea();     // invoke getArea()
      System.out.println(pizza.name + "'s area is " + area);

      Circle donut = new Circle();       // Circle object instantiation
      donut.radius = 2;                 // set radius
      donut.name = "자바도넛";           // set name
      area = donut.getArea();            // invoke getArea()
      System.out.println(donut.name + "'s area is " + area);
   }
}
```



Class: Field

- What can be the field?
  - Object's unique data
  - Object's current status
  - Sub-objects (components)
  - …
- Declaration
  - Same as variable declaration
  - Fields are initialized with the default values upon object instantiation, if no values were set



Class: Field (cont'd)

- Default values

Category		Type	Default value
Primitive	Integer	byte	0
		char	₩u0000
		short	0
		int	0
		long	0L
	Floating-point	float	0.0F
		double	0.0
	boolean	boolean	false
Reference		Array	null
		Class (including String)	null
		Interface	null

Class: Field (cont'd)

- How to access the fields?
  - How to read/write the contents of a field?
  - Inside an object
    - Direct access to a field by "fieldName"
  - Outside an object
    - Can access by "refVariable.fieldName"

```
public class Car { // Car class

    String company;
    String model;
    String color="red";
    int maxSpeed=150;

    public Car(){  // constructor
        company="SNUTECH";
        model="ITM";
    }

    public static void main(String[] args) {

        Car myCar = new Car();
        System.out.println(myCar.company);
        myCar.company="SeoulTech";
        System.out.println(myCar.company);

    }
}
```

Setting the values of the fields

Class: Constructor

- Who initializes an object?
  - Constructor!

- Constructor
  - Invoked when a new object is generated
  - Multiple definitions allowed
    - At least one constructor MUST be defined
  - Name of the constructor MUST be identical to the name of the class
    - ex) public className(…)
  - Cannot declare a return type
  - Can have arguments

```
public class Car { // Car class

    String company;
    String model;
    String color="red";
    int maxSpeed=150;

    public Car(){  // constructor
        company="SNUTECH";
        model="ITM";
    }                                   // initialization

    public static void main(String[] args) {

        Car myCar = new Car();
        System.out.println(myCar.company);
        myCar.company="SeoulTech";
        System.out.println(myCar.company);

    }
}
```

Constructor block

Class: Constructor (cont'd)

- Constructor with arguments
  - Arguments can be used for initialization
- Name conflict
  - Car's model field
  - Constructor's model argument
  - How to differentiate it?
- this keyword
  - Reference to a class/object itself

```java
public class Car { // Car class

    String company;
    String model;
    String color="red";
    int maxSpeed=150;

    public Car(String comp, String model){  // constructor
        company=comp;
        model=model;
    }

    public static void main(String[] args) {

        Car myCar = new Car( comp: "SeoulTech", model: "ITM");
        System.out.println(myCar.company);
        System.out.println(myCar.model);

    }
}
```

Class: Constructor (cont'd)

- Name conflict
  - Car's model field
  - Constructor's model argument
  - How to differentiate it?
- this keyword
  - Reference to a class/object itself
  - Use this.field syntax to represent a field of an object itself

```
public class Car { // Car class

    String company;
    String model;
    String color="red";
    int maxSpeed=150;

    public Car(String comp, String model){  // constructor
        company=comp;
        this.model=model;
    }

    public static void main(String[] args) {

        Car myCar = new Car( comp: "SeoulTech", model: "ITM");
        System.out.println(myCar.company);
        System.out.println(myCar.model);

    }
}
```

Class: Constructor (cont'd)

- Default constructor
  - Invoked when a new object is generated
  - At least one constructor MUST be defined
  - If no constructor is defined by developers, a compiler automatically inserts a default constructor

```
public class Circle {
   int radius;
   void set(int r) { radius = r; }
   double getArea() { return 3.14*radius*radius; }

   public static void main(String [] args){
      Circle pizza = new Circle();
      pizza.set(3);
   }
}
```

No constructor?
It's impossible!

```
public class Circle {
   int radius;
   void set(int r) { radius = r; }
   double getArea() { return 3.14*radius*radius; }

   public Circle() {}

   public static void main(String [] args){
      Circle pizza = new Circle();
      pizza.set(3);
   }
}
```

Class: Constructor (cont'd)

- Default constructor: exception
  - The default constructor is NOT added if a developer-defined constructor exists
  - Developer-defined constructor MUST be used when instantiating an object

```
public class Circle {
   int radius;
   void set(int r) { radius = r; }
   double getArea() { return 3.14*radius*radius; }


   public Circle(int r) {
      radius = r;
   }
   public static void main(String [] args){
      Circle pizza = new Circle(10);
      System.out.println(pizza.getArea());

      Circle donut = new Circle();
      System.out.println(donut.getArea());
   }
}
```

// Default constructor (i.e., public Circle()) is not defined

Class: Constructor (cont'd)

- Multiple constructors
  - Multiple constructors can be defined in a class
  - Various types of initialization can be performed

```
public class myClass {
   myClass (arguments, ...){
      ...
   }

   myClass (arguments, ...){
      ...
   }
```

[Constructor Overloading]
Same name, but the type and number of the arguments are different!

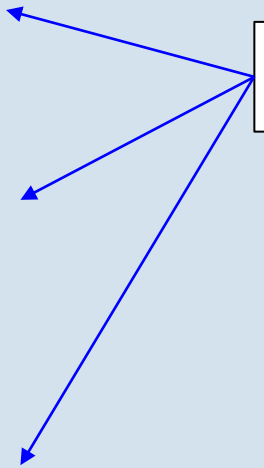
```
public class Car {

   Car(){ ... }
   Car(String model){ ... }
   Car(String model, String color) { ... }
   Car(String model, String color, int speed) { ... }
}
```

```
Car car1 = new Car();

Car car2 = new Car("WowCar");

Car car3 = new Car("WowCar", "gold");

Car car4 = new Car("WowCar", "gold", 300);
```

Class: Constructor (cont'd)

- Multiple constructors
  - Some codes may be duplicated in the multiple constructors

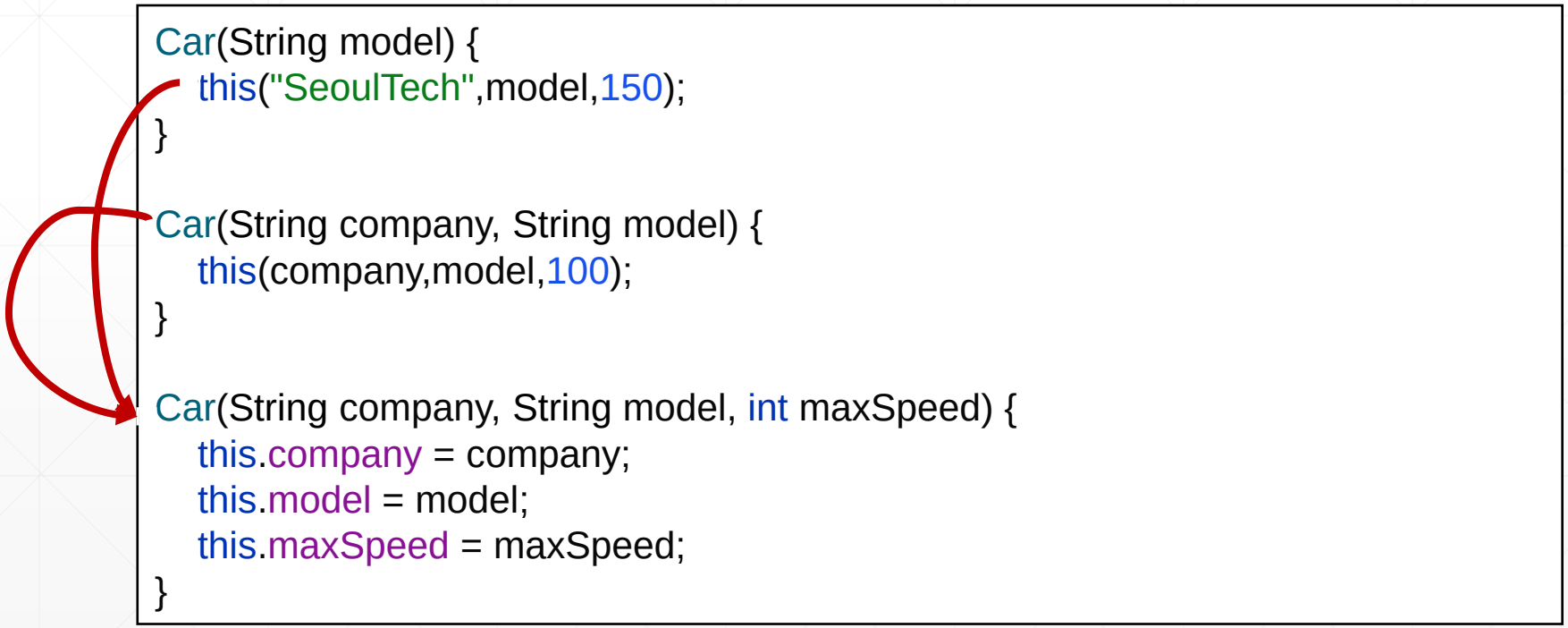
```
Car (String model){
   this.company = "SeoulTech";
   this.model = model;
   this.maxSpeed=100;
}

Car (String company, String model){
   this.company = company;
   this.model = model;
   this.maxSpeed=100;
}


Car (String company, String model, int maxSpeed){
   this.company = company;
   this.model = model;
   this.maxSpeed=maxSpeed;
}
```



[Constructor Overloading]
Same name, but the type and number of the arguments are different!

Class: Constructor (cont'd)

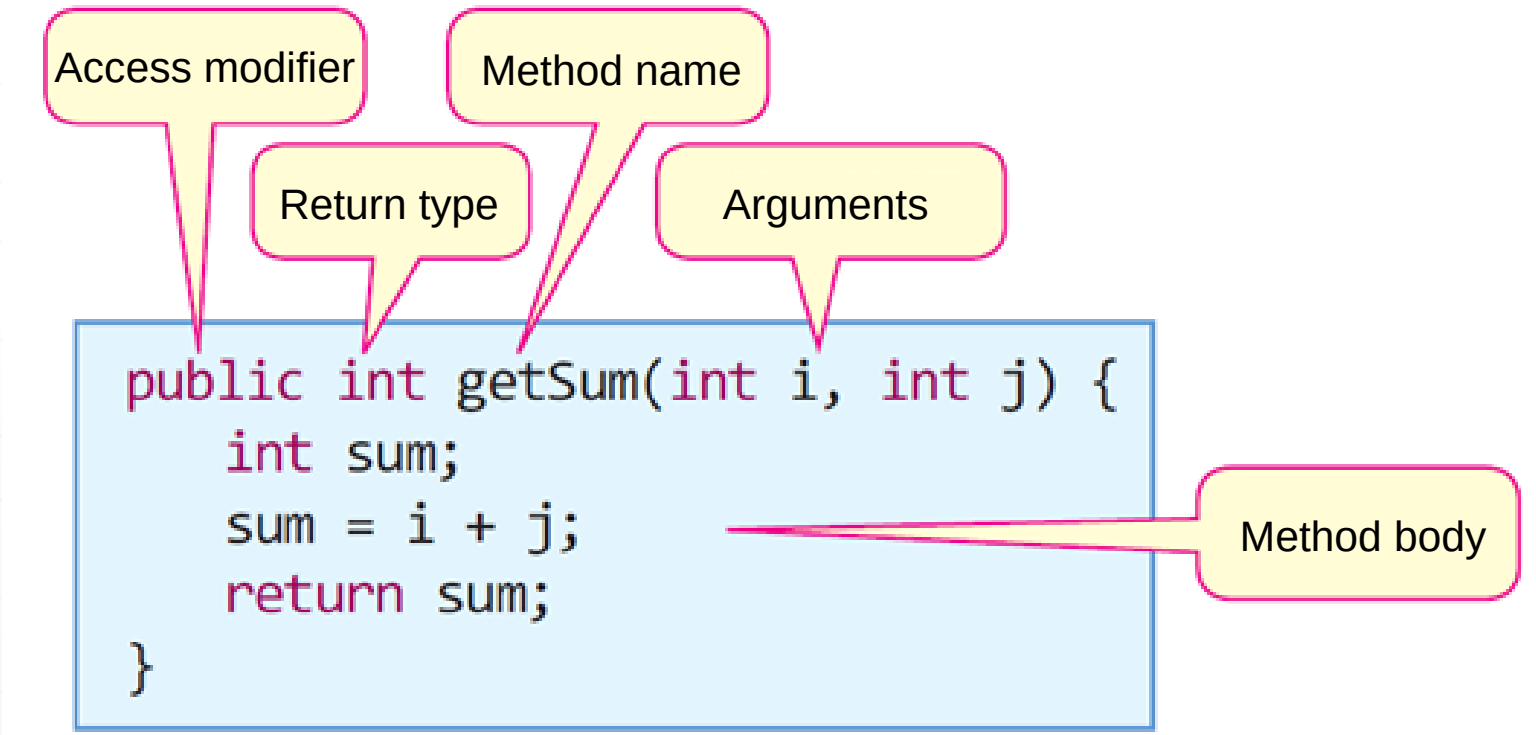
- Multiple constructors
  - this() can be used for calling another constructor in the class
  - this() MUST be used in the first line of a constructor



```
Car(String model) {
    this("SeoulTech",model,150);
}

Car(String company, String model) {
    this(company,model,100);
}

Car(String company, String model, int maxSpeed) {
    this.company = company;
    this.model = model;
    this.maxSpeed = maxSpeed;
}
```

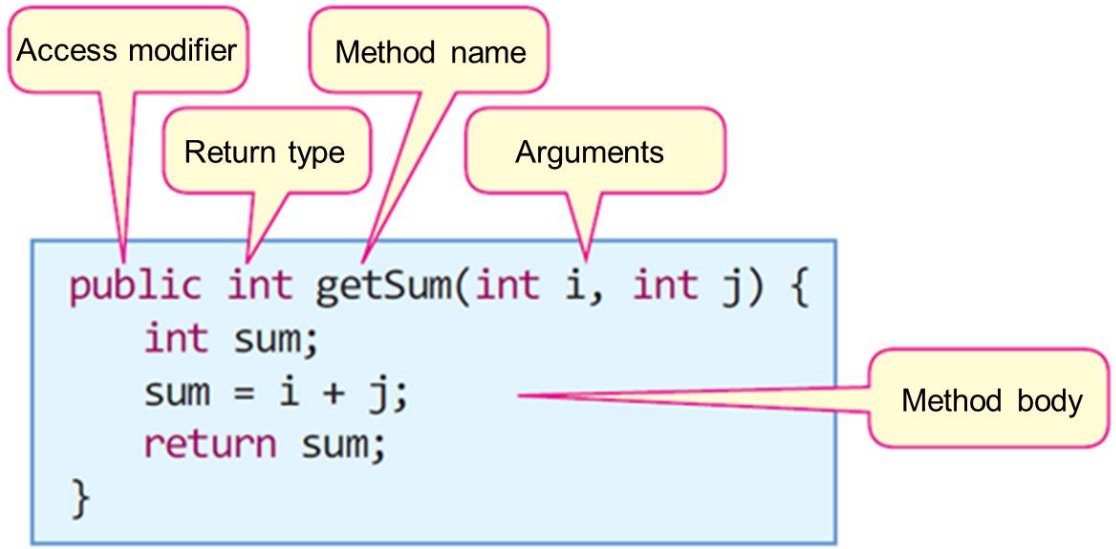
Class: Method

- Behavior of a class
  - Behavior of a class is implemented through a method



Class: Method (cont'd)

- Access modifier
  - public, private, protected, default
- Return type
  - The type of the data returned by a method
  - Method can have no return values (void)
- Method name
  - Method name should follow the JAVA naming rule
- Arguments
  - Input data for a method
  - Method can have no input values



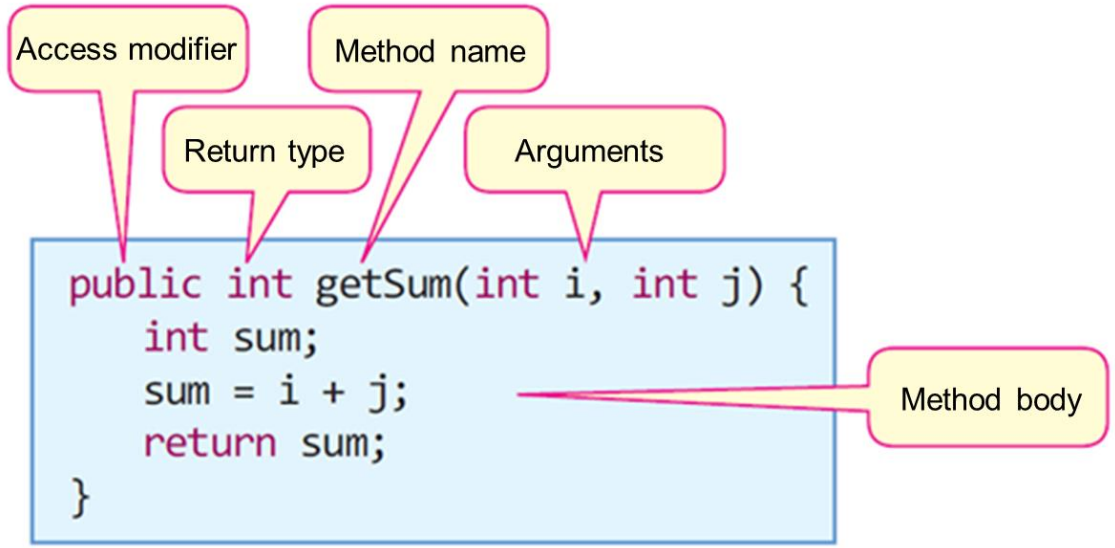

```
void powerOn() { ... }
double divide(int x, int y) { ... }
```
// method implementation

```
powerOn();
double result = divide( 10, 20 );
```
// method call

```
byte b1 = 10;
byte b2 = 20;
double result = divide(b1, b2);
```
// method call

Class: Method (cont'd)

- Return statement
  - Method with a return type
    - Terminates the method and returns a value
    - The statements after return is not reachable
  - Method without a return type
    - Terminates the method
    - The statements after return is not reachable



Class: Method (cont'd)

- Return statement
  - Method with a return type
    - Terminates the method and returns a value
    - The statements after return is not reachable
  - Method without a return type
    - Terminates the method
    - The statements after return is not reachable

```
public double getSpeedByMile(){
    return maxSpeed * 0.62137;
}

public void showInfo(){
    if(maxSpeed<100) return;
    System.out.println(company+"_"+model);
    System.out.println(maxSpeed);
}

public static void main(String[] args) {

    Car myCar = new Car("SeoulTech", "ITM", 99);
    double mySpeed = myCar.getSpeedByMile();
    System.out.println(mySpeed);
    myCar.showInfo();

}
```

Class: Method (cont'd)

Method invocation

- Method call inside a class
  - Call a method via its name

- Method call outside a class
  - After creation of a class (i.e., object instantiation)
  - Call the method through a reference (obj.method)

```
public double getSpeedByMile(){
    return maxSpeed * 0.62137;
}

public void showInfo(){
    if(maxSpeed<100) return;
    System.out.println(company+"_"+model);
    System.out.println(getSpeedByMile());
}

public static void main(String[] args) {

    Car myCar = new Car("SeoulTech", "ITM", 99);
    double mySpeed = myCar.getSpeedByMile();
    System.out.println(mySpeed);
    myCar.showInfo();

}
```

Class: Method (cont'd)

Argument passing

- Passing primitive-type values
  - A value is copied and then passed to the method
  - Change of the argument does not affect the original value

- Passing reference-type values (e.g., object, array, etc.)
  - A reference is passed to the method
  - Change of the argument affects the original value

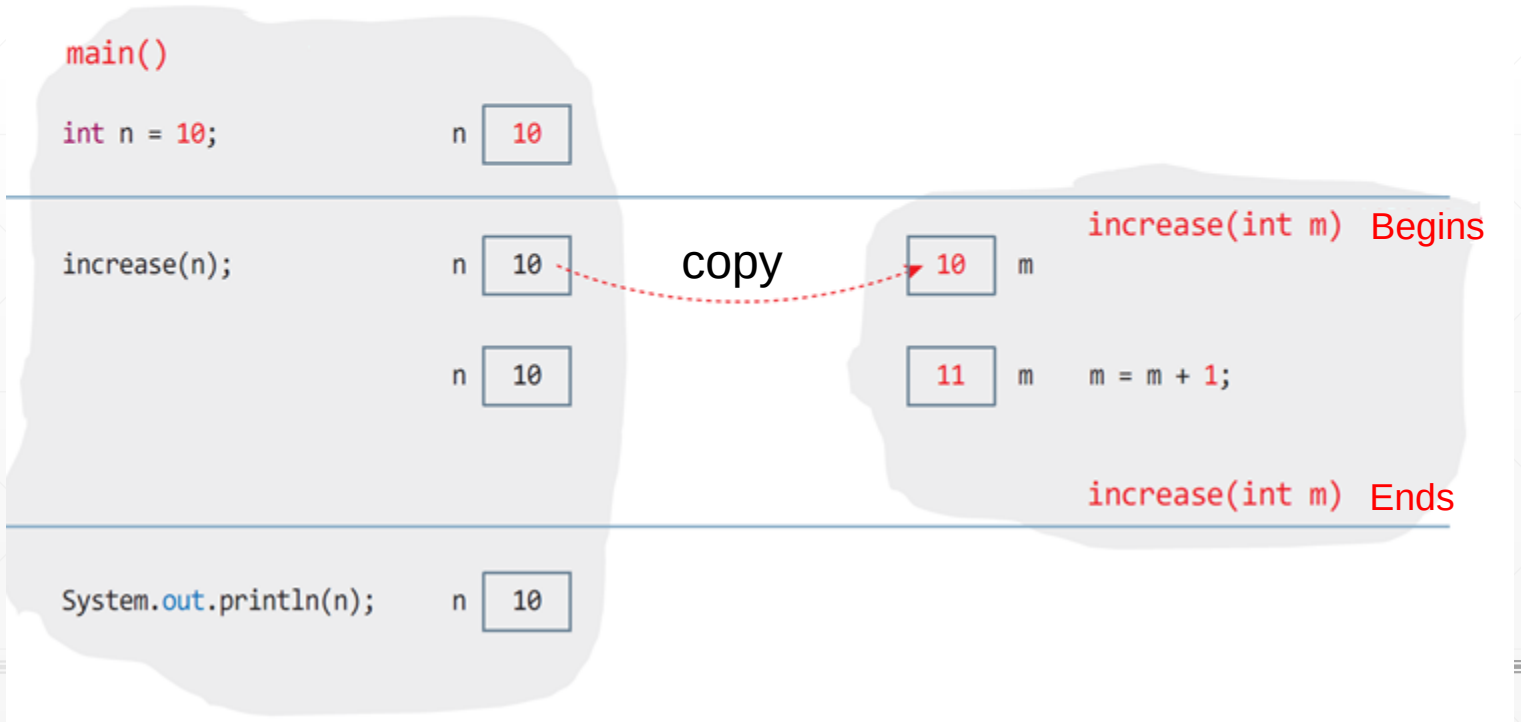
Class: Method (cont'd)

Argument passing (Passing primitive-type values)

- A value is copied and then passed to the method
- Change of the argument does not affect the original value

```
public class ValuePassing {
   public static void main(String args[]) {
      int n = 10;

      increase(n);

      System.out.println(n);
   }
```

```
   static void increase(int m) {
      m = m + 1;
   }

}
```



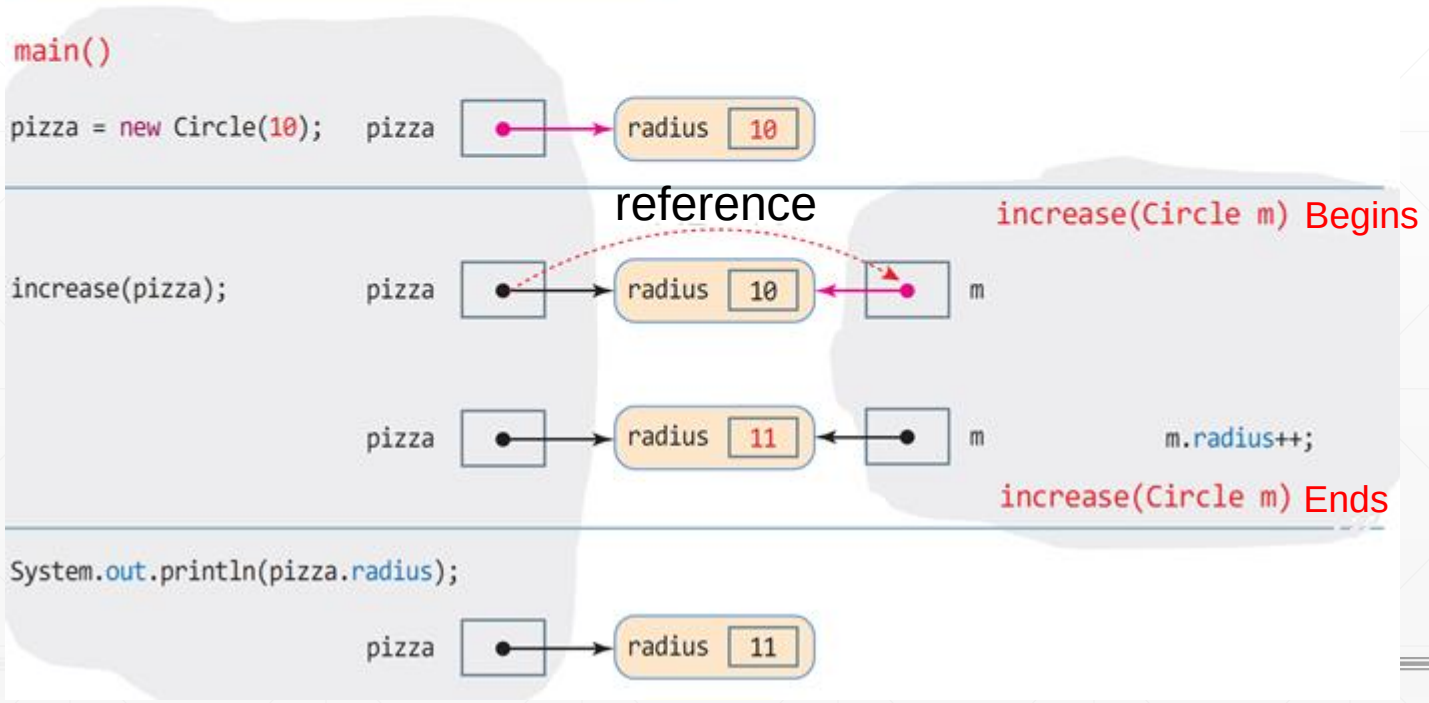
Class: Method (cont'd)

Argument passing (Passing reference-type values)

- A reference is passed to the method
- Change of the argument affects the original value

```
public class ReferencePassing {
   public static void main (String args[]) {
      Circle pizza = new Circle(10);

      increase(pizza);

      System.out.println(pizza.radius);
   }
```

```
   static void increase(Circle m) {
      m.radius++;
   }

}
```

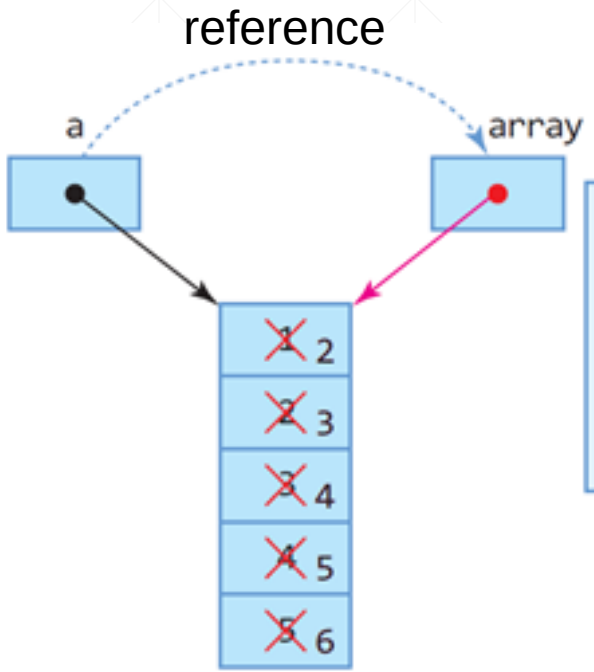


Class: Method (cont'd)

Argument passing (Passing reference-type values)

- A reference is passed to the method
- Change of the argument affects the original value

```
public class ArrayPassing {

    public static void main(String args[]) {
        int a[] = {1, 2, 3, 4, 5};


        increase(a);


        for(int i=0; i<a.length; i++)
            System.out.print(a[i]+" ");
    }
```




```
    static void increase(int[] array) {
        for(int i=0; i<array.length; i++) {
            array[i]++;
        }
    }
}
```

Class: Method Overloading

- Methods with the same name, but with the different number/type of arguments

```
// successful method overloading

class MethodOverloading {
  public int getSum(int i, int j) {
    return i + j;
  }
  public int getSum(int i, int j, int k) {
    return i + j + k;
  }
}
```

```
// Fail!

class MethodOverloadingFail {
  public int getSum(int i, int j) {
    return i + j;
  }
  public double getSum(int i, int j) {
    return (double)(i + j);
  }
}
```

Class: Method Overloading (cont'd)

- Methods with the same name, but with the different number/type of arguments

```
public static void main(String args[]) {
    MethodSample a = new MethodSample();

    int i = a.getSum(1, 2);

    int j = a.getSum(1, 2, 3);

    double k = a.getSum(1.1, 2.2);

}
```

3 different method calls

```
public class MethodSample {
    public int getSum(int i, int j) {
        return i + j;
    }

    public int getSum(int i, int j, int k) {
        return i + j + k;
    }

    public double getSum(double i, double j) {
        return i + j;
    }
}
```

Q&A

- Next week (eClass video)
  - OOD/P: More about Methods
  - OOD/P: Inheritance