ROAD-TO-NINJA

Core Java - Beginner (Part 2)

Object Oriented (Part 1)

Organised by :

Supported by :

ABOUT ME

Name :	**Mohd Azman Kudus**
Age :	30 years
Java exp :	7 years

Question?

AGENDA

- Programming paradigm
- Object Oriented (OO) paradigm
  - Purpose
  - History
- Analysis (OOA)
  - Entity, characteristic, data
  - Object modelling (OOM)
  - Unified Modelling Language
    - Use case diagram
    - Class diagram

AGENDA

- Design (OOD)
  - Applies into skeleton/pseudo code
  - Concept design
  - Constraints evaluation

- Programming (OOP)
  - Composition
  - Inheritance
  - Method overload & override
  - Polymorphism
  - Abstraction
  - Interface

PROGRAMMING PARADIGMS

- A *programming paradigm* is a style, or "way," of programming.

- Some languages make it easy to write in some paradigms but not others.

Never use the phrase "programming language paradigm."

A paradigm is a way of doing something (like programming), not a concrete thing (like a language).

PROGRAMMING PARADIGMS

REFLECTIVE

FUNCTIONAL

ARRAY

CONSTRAINT

DECLARATIVE

LOGIC

OBJECT-ORIENTED

PROCEDURAL

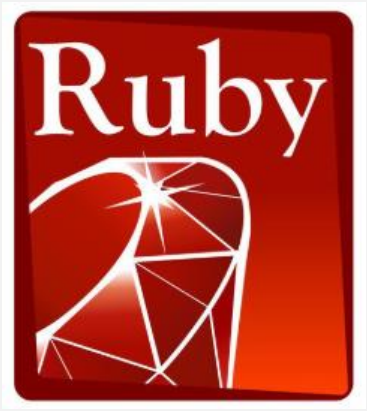
EVENT-DRIVEN

FLOW-DRIVEN

ASPECT-ORIENTED

IMPERATIVE

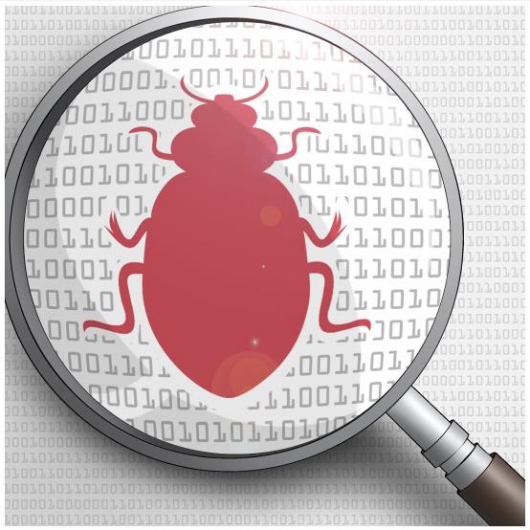
PAST - PRESENT



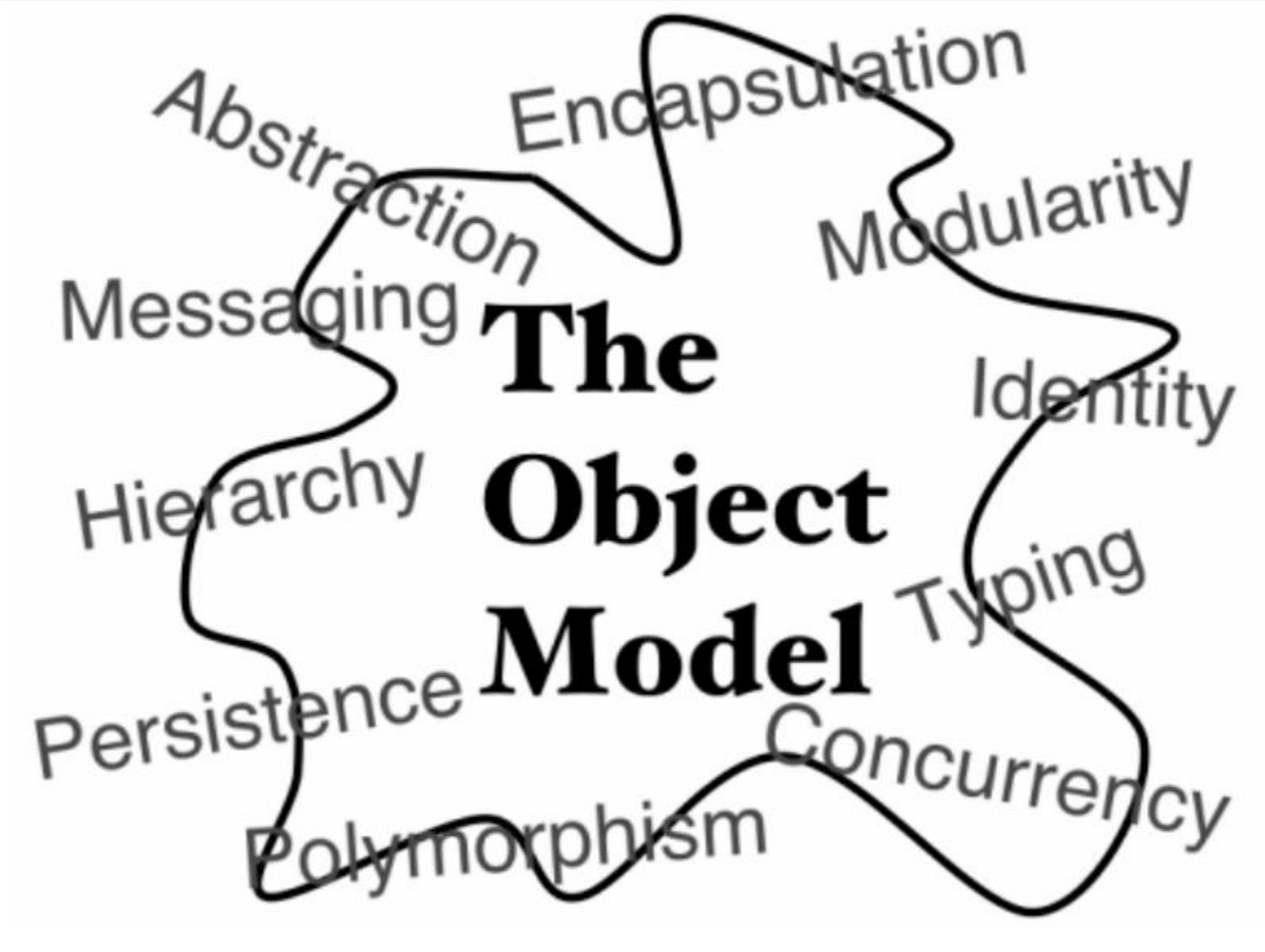
MOTIVATION

- Solve problems in complex domain.
- Code and design reusability
- Evolve over time
- Maintenance without design restructuring.
- Lower risk and avoid software crisis
- Resembles human cognition

SOFTWARE CRISIS



OBJECT


Abstraction
Encapsulation
Modularity
Messaging
The
Object
Model
Identity
Hierarchy
Typing
Persistence
Concurrency
Polymorphism

ANALYSIS (OOA)

DEFINE real problem and
what should we/software do

DESCRIBE requirements

MODEL the solutions
(Objects, processes, flows)

ANALYSIS (OOA)

Object	Entity Human A thing
State	Characteristics Property
Behavior	Action Process

I have a friend named Joe.

He is 5 feet 7 inch tall

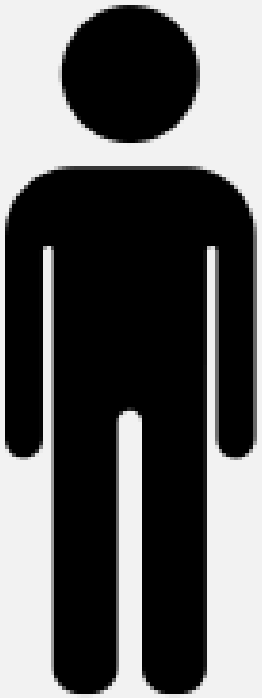
and 150 pounds weight.

Can you help to determine his BMI?

OOA - IDENTIFICATION

Friend

Object / Entity



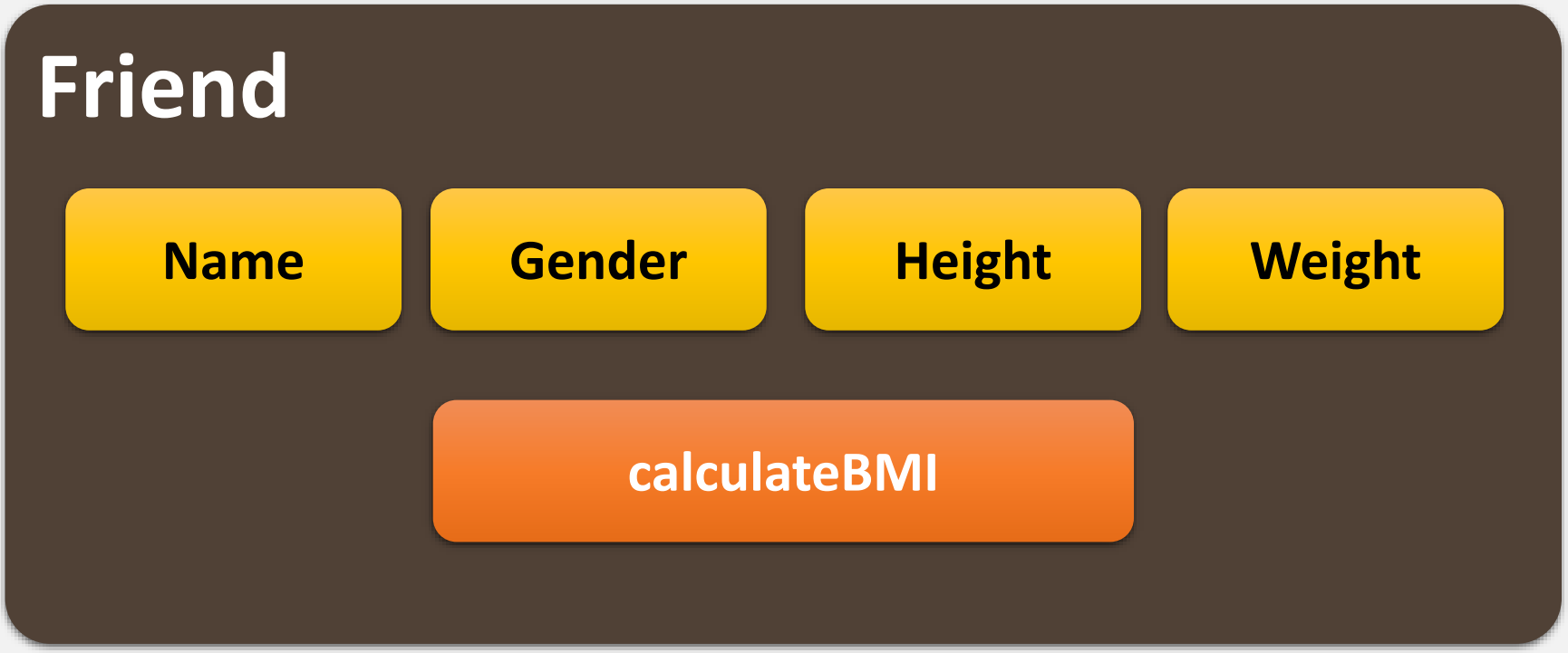
Name	Joe
Gender	Male
Height	5' 7"
Weight	150 lb
BMI	23.49

Pre-defined data (Field)

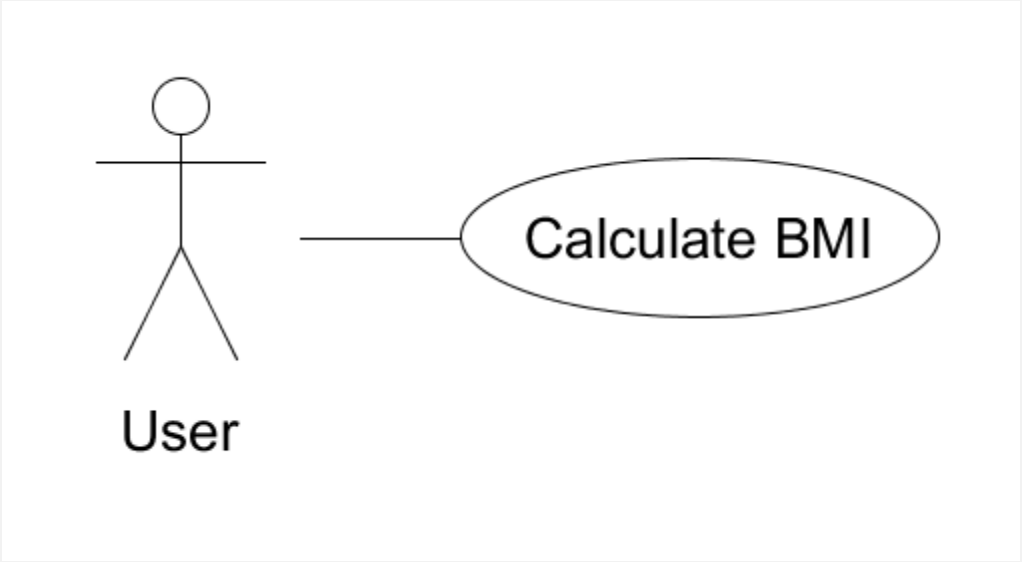
Derived data produced by a process (Method)

Characteristics

OOA - MODEL



Use Case Diagram



UNIFIED MODELLING LANGUAGE

by Grady Booch, Ivar Jacobson and James Rumbaugh (1994)

A standard way to visualize the design of a system.

Diagrams

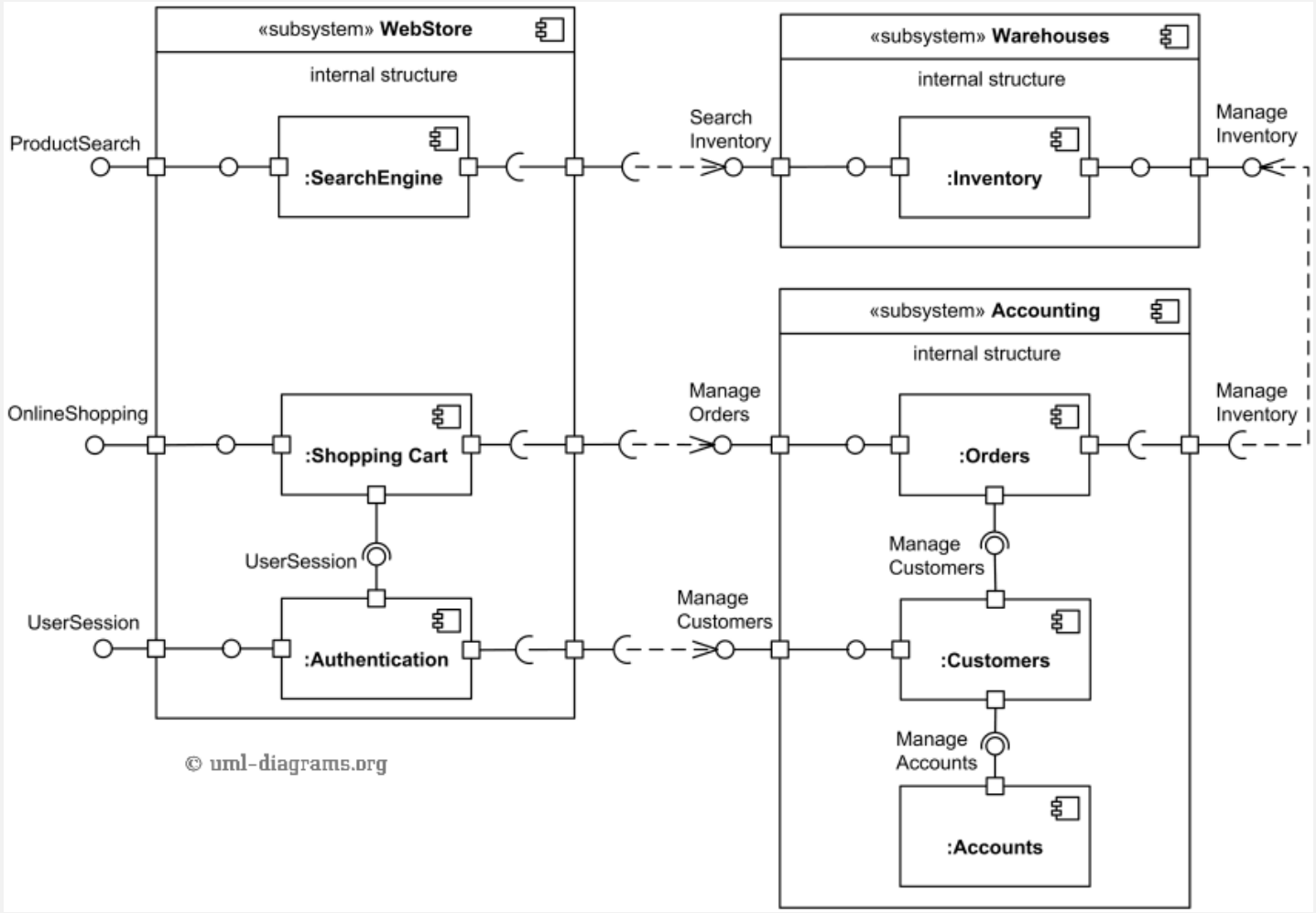
Structure (Component, Class)

Behavior (Activity, Use Case)

Interaction (Sequence, Communication)

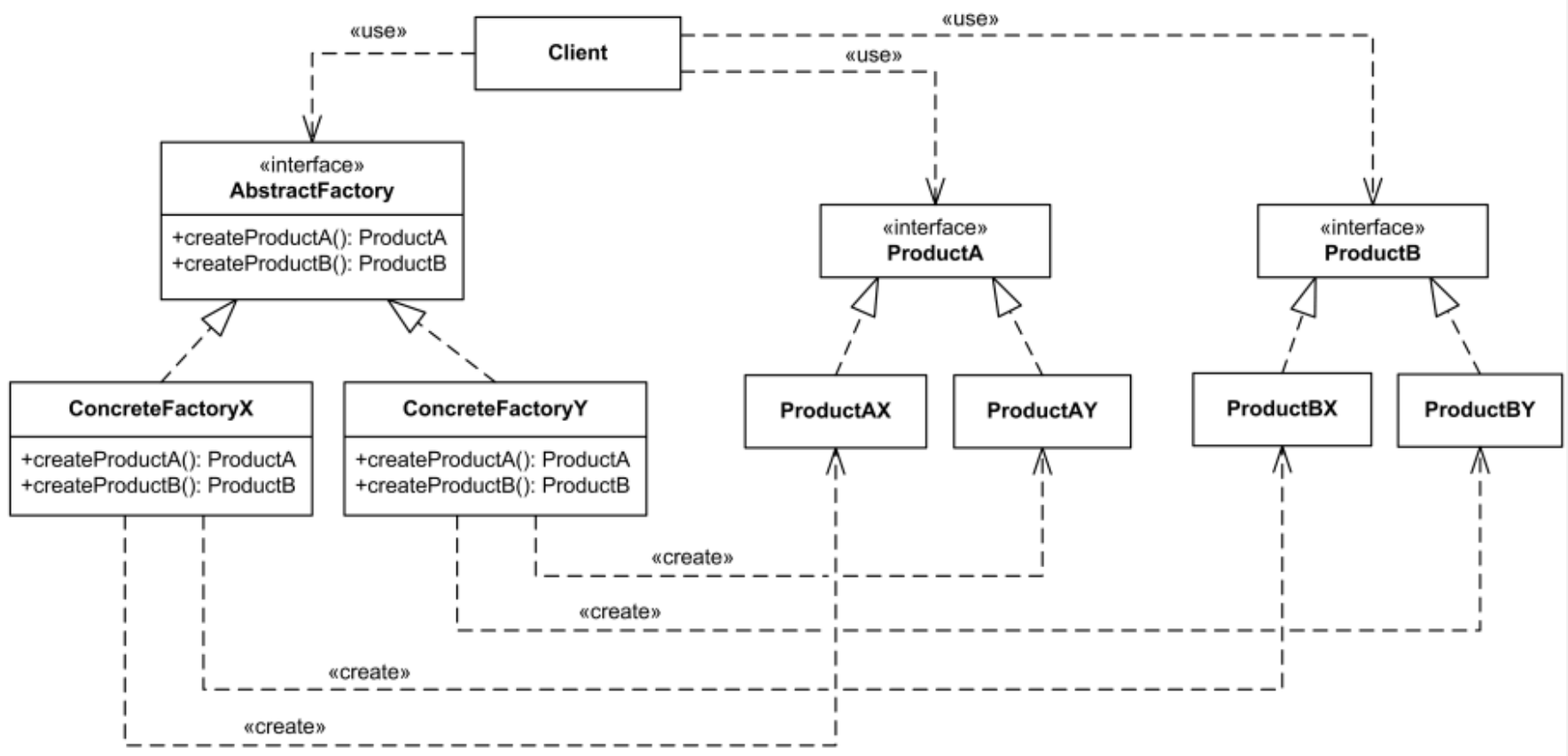
UML - COMPONENT DIAGRAM

- Shows components and dependencies between them.
- Usually used for :
  - **Component-Based Development (CBD)**
  - **Service-Oriented Architecture (SOA)**



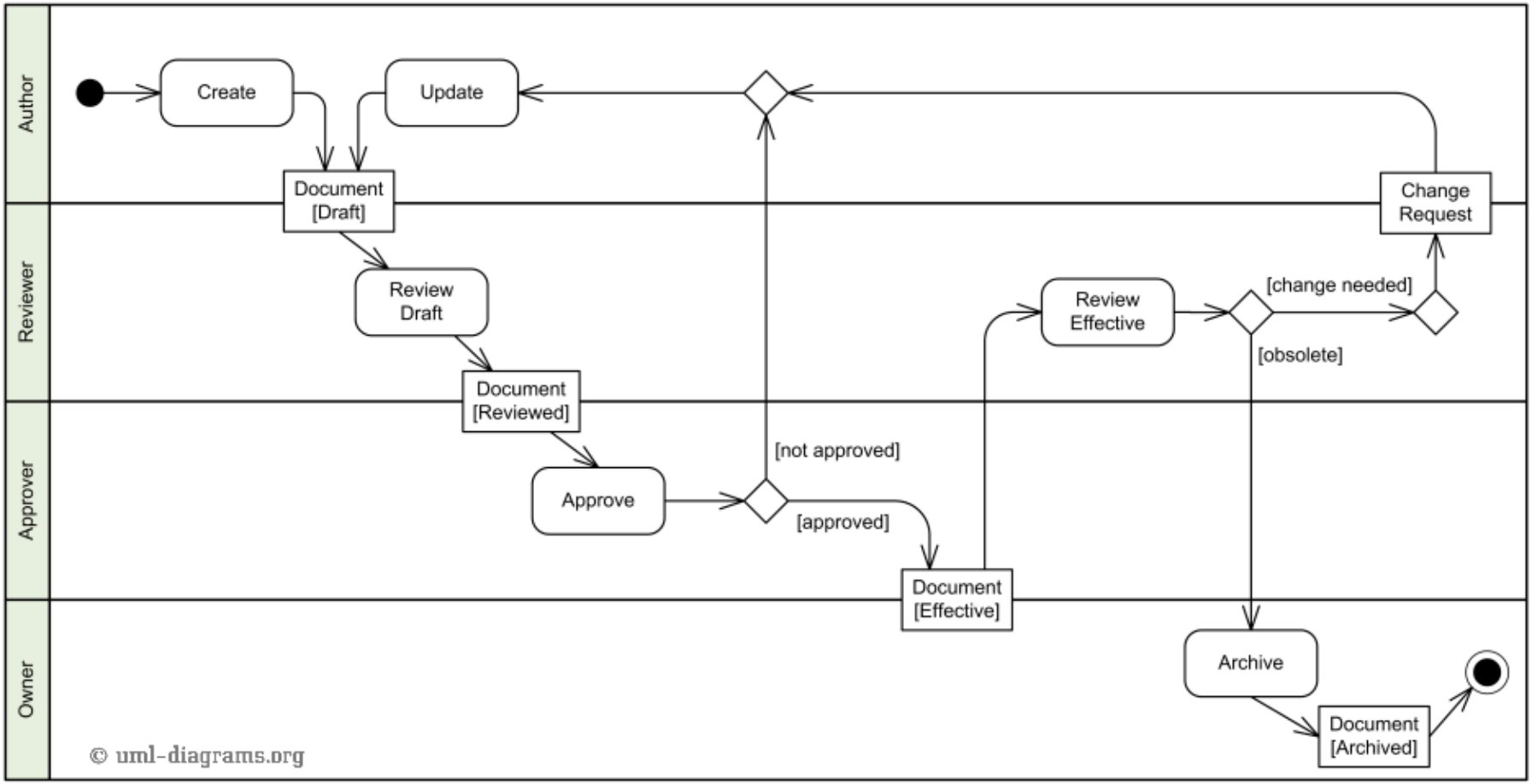
UML - CLASS DIAGRAM

- Shows structure of the designed system, subsystem or component as related classes and interfaces
- May contains features, constraints and relationships (associations, generalizations, dependencies)



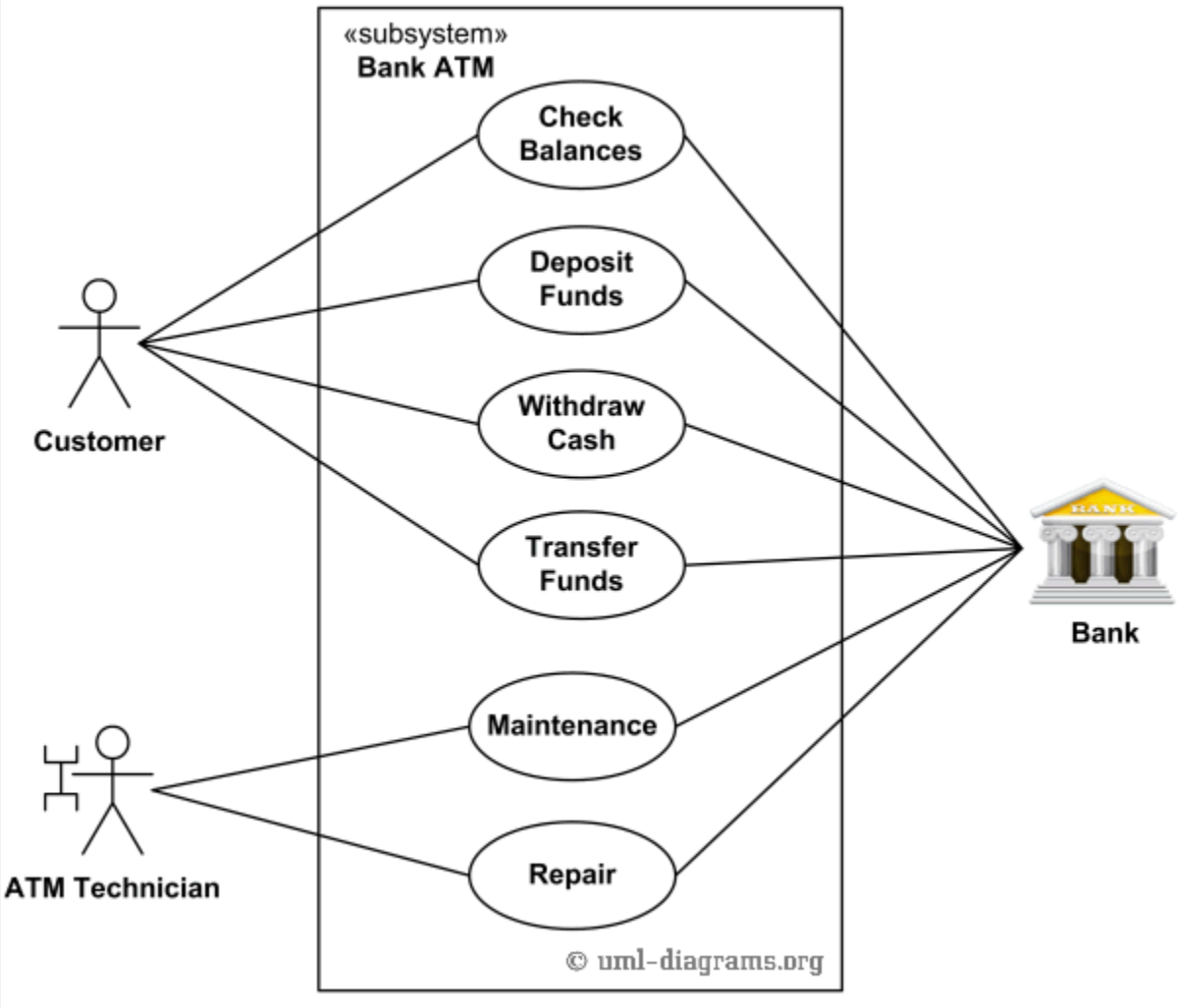
UML - ACTIVITY DIAGRAM

Shows sequence and conditions for coordinating lower-level behaviors
Control flow and object flow



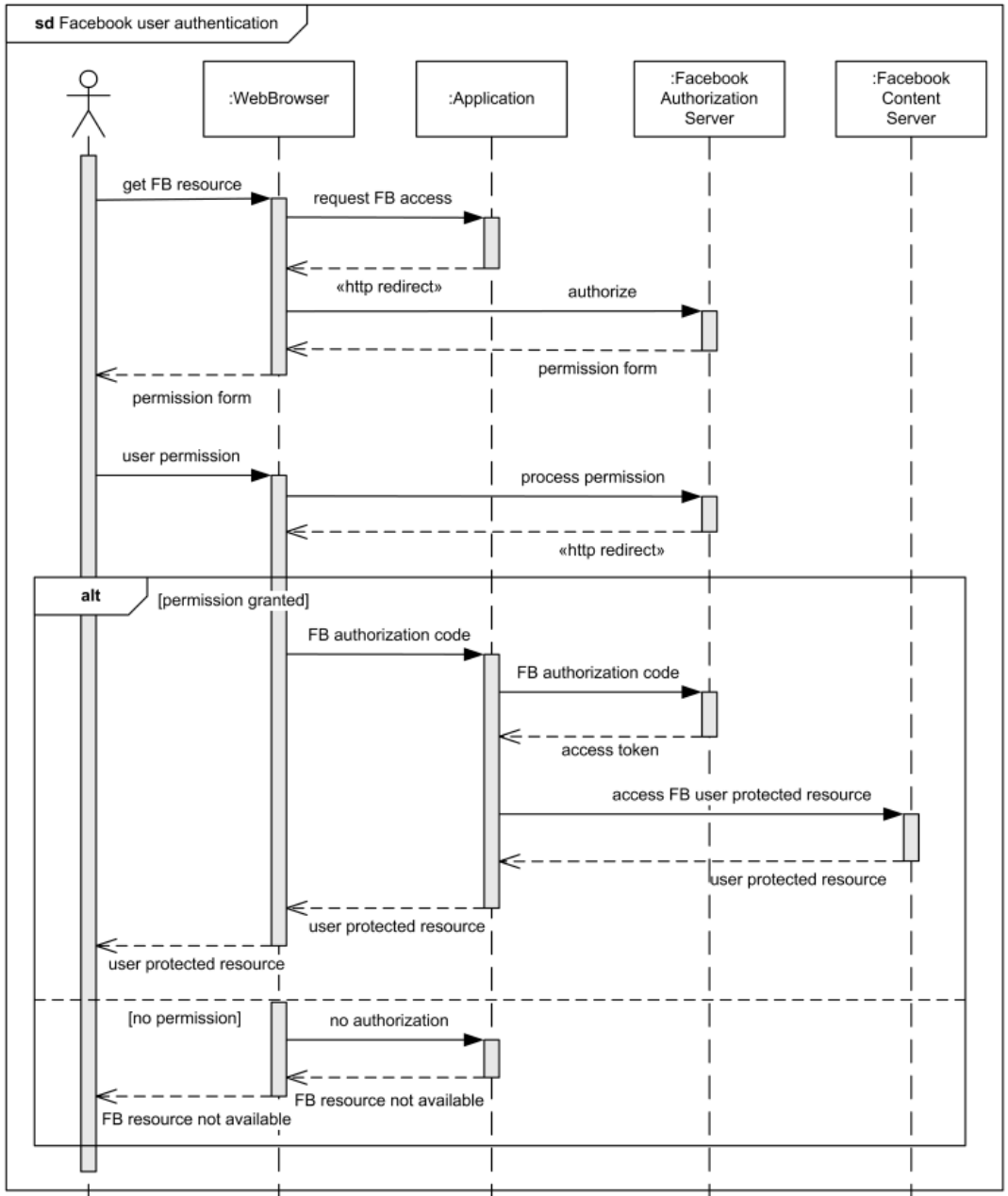
UML - USE CASE DIAGRAM

Describe actions can be performed by stakeholders



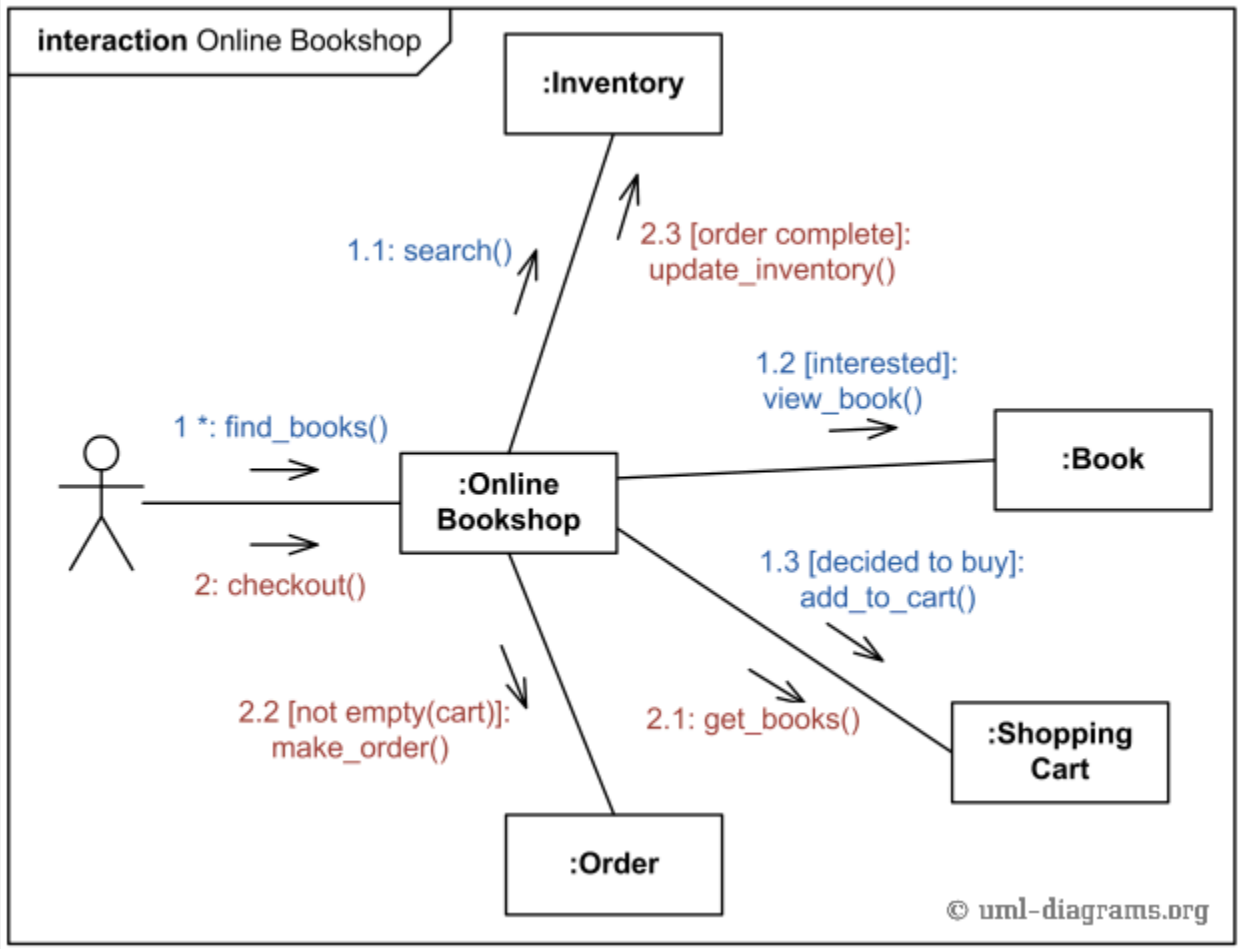
UML - SEQUENCE DIAGRAM

Focuses on the message interchange between lifelines (objects)



UML - COMMUNICATION DIAGRAM

Focuses on the message interchange between lifelines' internal structures.

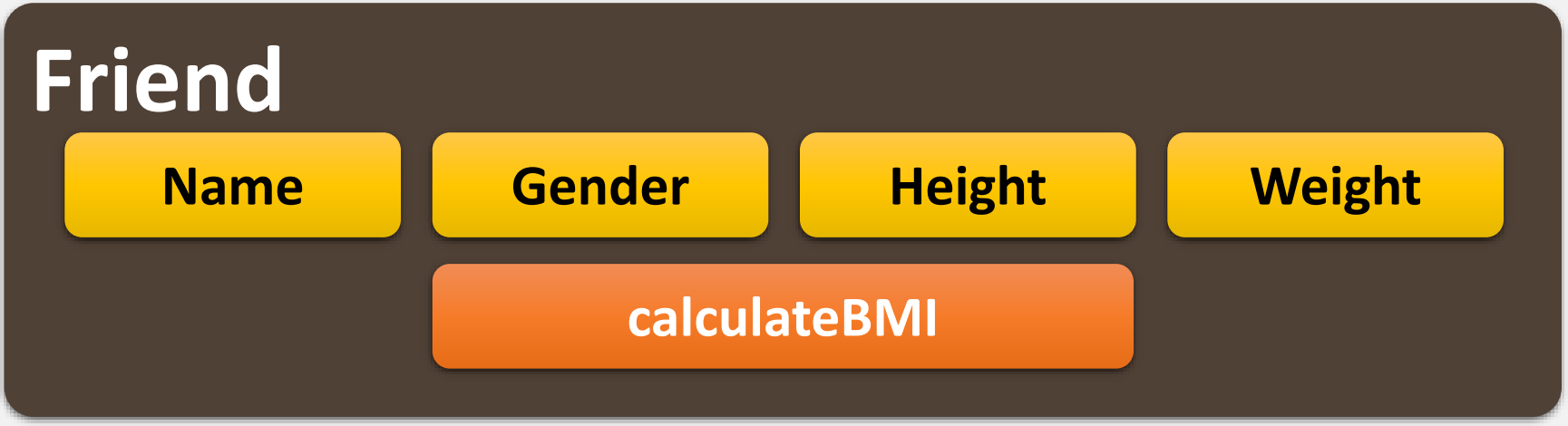


DESIGN (OOD)

DESCRIBE the solutions
(objects and interactions)

APPLY software design
principles / patterns

OOD - IDENTIFICATION



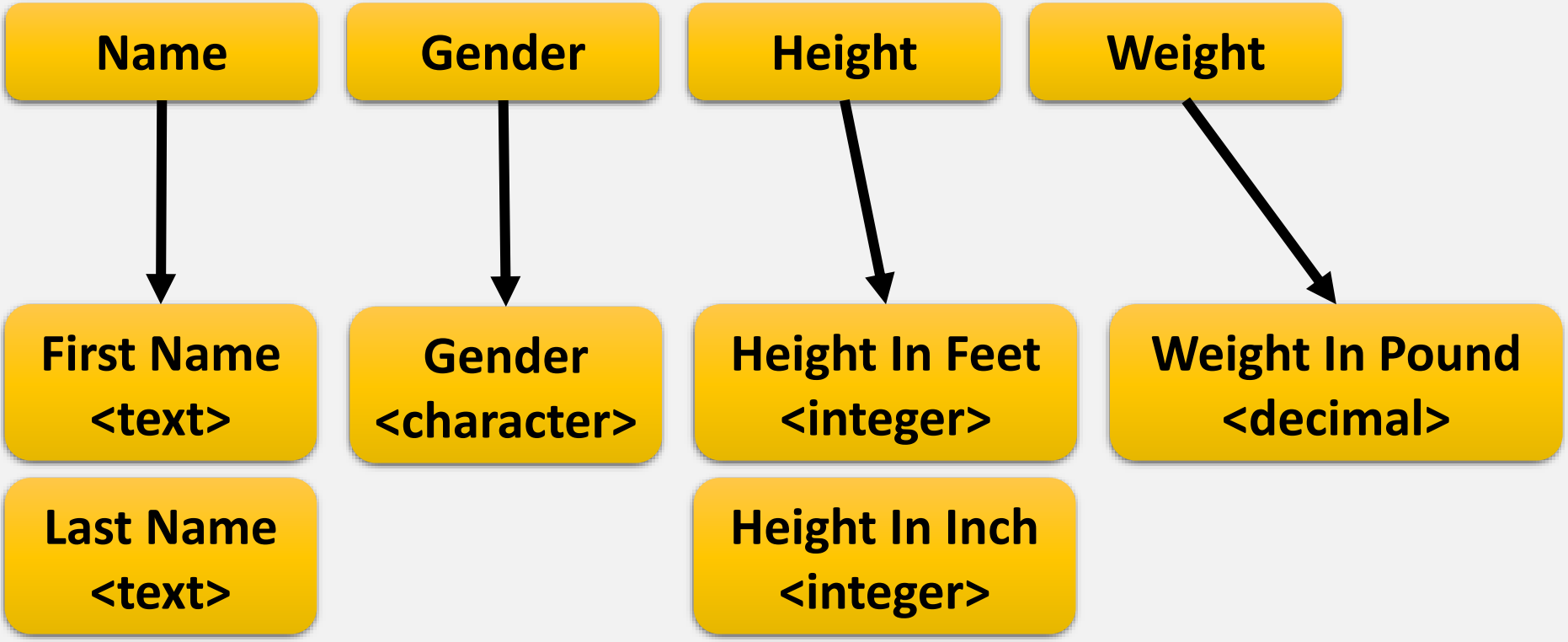
Data type and limit

Data separation

Data conversion process

Formula and output

OOD - BREAKDOWN



OOD - BREAKDOWN

calculateBMI

$$BMI = \frac{\text{weight (kg)}}{\text{height (m}^2\text{)}}$$

Height In Feet <integer>

Height In Inch <integer>

Weight In Pound <decimal>

Get Height In Meter
Convert Feet to Inch
Convert Inch to Meter

Get Weight In Kilogram
Convert Pound to Kilogram

OOD - BREAKDOWN

Convert Feet to Inch

1 feet = 12 inch

Convert Inch to Meter

1 inch = 0.0254 m

Convert Pound to Kilogram

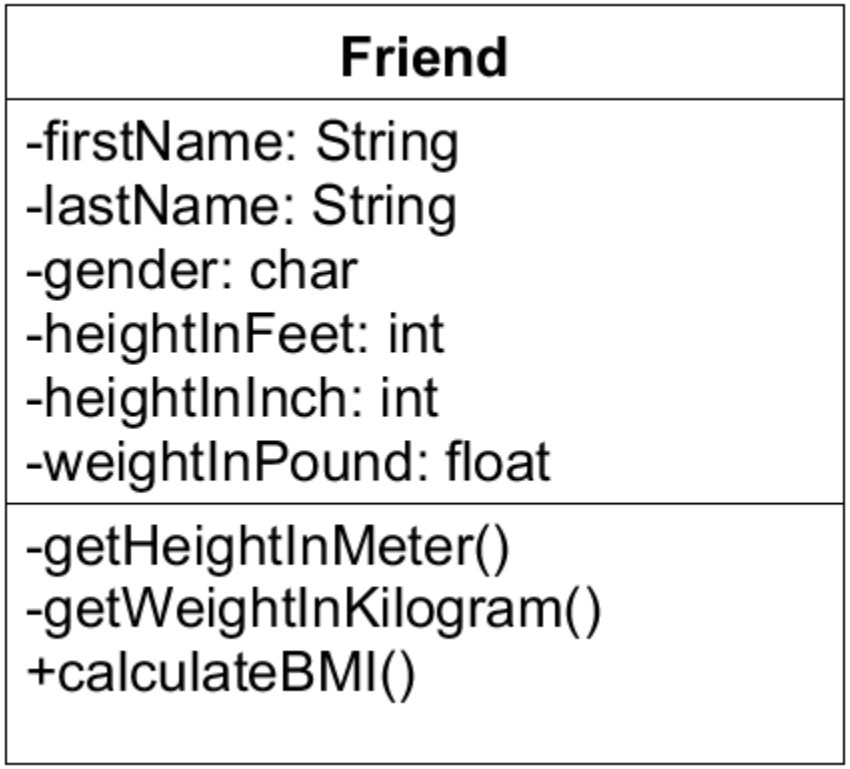
1 lb = 0.453592 kg

Friend

- FirstName
- Gender
- HeightInFeet
- WeightInPound
- LastName
- HeightInInch

- GetHeightInMeter
- GetWeightInKilogram
- CalculateBMI

Class Diagram



OOD - PSEUDO

```
class Friend {
    variables

    normal constructor;

    get height in meter;
    get weight in kilogram;

    get BMI;
}
```

EXERCISE #1

Calculate circle area (cm^2) and perimeter (cm) of a circle with given diameter in inch.

PROGRAMMING (OOP)

CLASS

ENCAPSULATION

INHERITANCE

POLYMORPHISM

ABSTRACTION

OOP - CLASS

Blueprint of an object

```
public class ClassName {
    ...
}
```

- ✓Field
- ✓Constructor
- ✓Accessor
- ✓Mutator
- ✓Supplementary methods

CLASS - FIELD

Field is a reference to storage which hold input/output/static/derived values/instances.

```
public class ClassName {

    public static final String CONSTANT = "value";

    public String variable1;
    public int variable2;

    ...
}
```

STATIC VALUE / CONSTANT

INSTANCE VALUE / VARIABLE

CLASS - CONSTRUCTOR

Way to create new object instance. No return data type unlike methods.

1. Default constructor = no parameter (default if no constructor defined)
2. Normal constructor = with parameters

```
public class ClassName {
    ...

    public ClassName() {
        // initialisation
    }

    public ClassName(String var1, int var2) {
        this.var1 = var1;
        this.var2 = var2;
        // initialisation
    }

    ...
}
```

DEFAULT CONSTRUCTOR

NORMAL CONSTRUCTOR

CLASS - ACCESSOR

A method to get values from an object.
Usually called getter and method name starts with "get".
Avoid external classes from controlling the variable

```
public class ClassName {

    private String var1;
    private int var2;

    ...

    public String getVar1() {
        return var1;
    }

    public int getVar2() {
        return var2;
    }
}
```

PRIVATE VARIABLES

ACCESSOR / GETTER METHOD

CLASS - MUTATOR

A method to set values from an object.
Usually called setter and method name starts with "set".
Provide another layer of control/validation before update the variable.

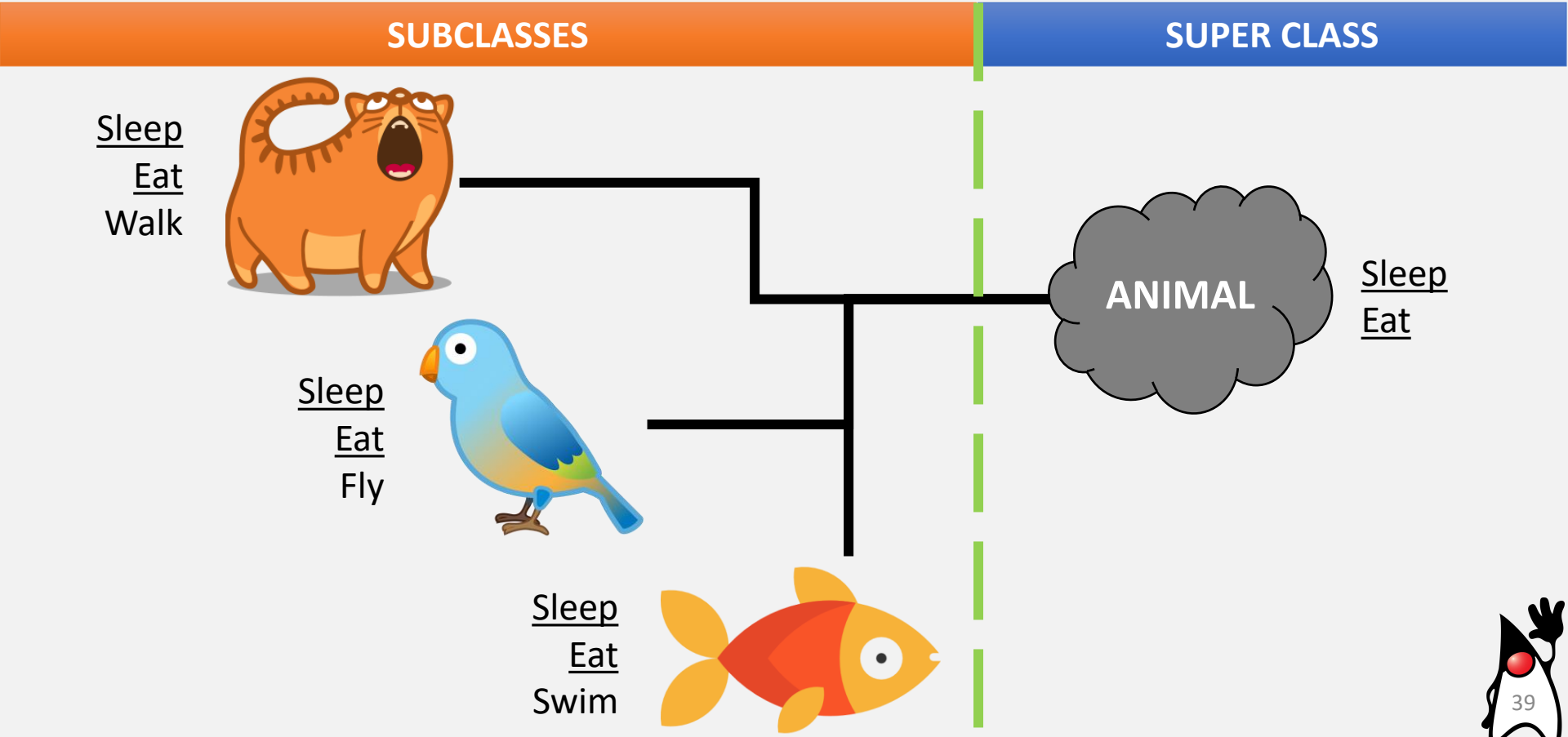
```
public class ClassName {

    private String var1;
    private int var2;

    ...

    public void setVar1(String var1) {
        // validation
        this.var1 = var1;
    }
}
```

PRIVATE VARIABLES

MUTATOR / SETTER METHOD

OOP - INHERITANCE

- ✓ Several objects may contains similar states or behaviors.
- ✓ We put these similarities into a common object/class called parent/super class.
- ✓ Then, the objects that having the similarities may inherit from superclass. These objects are called child/sub class.



SUPERCLASS

- ✓ Superclass will have the similarities between objects

```
public class Animal {
    ...

    public void sleep() {
        System.out.println("Sleep");
    }

    public void eat(String food) {
        System.out.println("Eat " + food);
    }
}
```

SUBCLASSES

```
public class Cat extends Animal {
    ...

    public void walk() {
        System.out.println("Walk");
    }
}
```

```
public class Bird extends Animal {
    ...

    public void fly() {
        System.out.println("Fly");
    }
}
```

"extends" KEYWORD TO INHERIT FROM ANOTHER OBJECT

```
public class Fish extends Animal {
    ...

    public void swim() {
        System.out.println("Swim");
    }
}
```

OVERLOADING

- ✓ Overloading is an alternative to create a method with same name as existing method with different parameters.

```
public class Animal {
    ...

    public void eat(String food) {
        System.out.println("Eat " + food);
    }

    public void eat(int times) {
        System.out.println("Eat " + amount + " times");
    }

    public void eat(String food, int amount) {
        System.out.println("Eat " + food +
            " " + amount + " times");
    }
}
```

Different data type

Additional parameter

OVERRIDING

- ✓ Overriding is a way for subclass to create/implement same method as in the superclass.
- ✓ Hence, when a subclass instance invoke the method, it will invoke the subclass method.

```
public class Animal {
    ...

    public void sleep() {
        System.out.println("Sleep");
    }
}
```

Parent method

```
public class Cat extends Animal {
    ...

    public void sleep() {
        System.out.println("Sleep in a box");
    }
}
```

Child method which override parent method

OVERRIDING


```
public class Cat extends Animal {
    ...

    @Override
    public void sleep() {
        System.out.println("Sleep in a box");
    }
}
```

"Override" keyword/annotation to indicate this method override parent method

INVOKE SUBCLASS

- ✓ Subclasses can override parent state or behaviors, sometimes it will be confusing whether invoke from superclass or subclass.
- ✓ Keyword "this"
- ✓ By default, all invocation will be from superclass except for overridden state or behaviors;

```
this.field;

this.method();

this.method(parameter);
```

Invoke current class field

Invoke current class method

Invoke current class method with parameter

INVOKE SUPERCLASS

- ✓ Subclasses can invoke state or behaviors from superclass
- ✓ Keyword "super"
- ✓ By default, all super class state or behaviors can be access directly by subclasses, without "super" keyword. Except for overridden methods.

```
super.field;

super();

super(parameter);

super.method();

super.method(parameter);
```

Invoke superclass field

Invoke superclass default constructor

Invoke superclass normal constructor

Invoke superclass method

Invoke superclass method with parameter

THIS & SUPER

```
public class Cat extends Animal {
    private String name = "Amy";

    public Cat() {
        super();
    }

    public Cat(String name) {
        super(name);
    }

    public void sleep() {
        System.out.println("Sleep in a box");
    }

    public void print() {
        System.out.println(this.name);
        System.out.println(super.name);
        System.out.println(name);
        System.out.println(super.age);
        System.out.println(age);
        this.sleep();
        super.sleep();
        sleep();
        eat("sausage");
    }
}
```

Invoke superclass default constructor

Invoke superclass normal constructor

Invoke current class field

Invoke superclass field

Invoke current class method

Invoke superclass method

IS-A

Any subclass IS-A superclass

```
public class Cat extends Animal {
    ...
}
```

Cat IS-A Animal

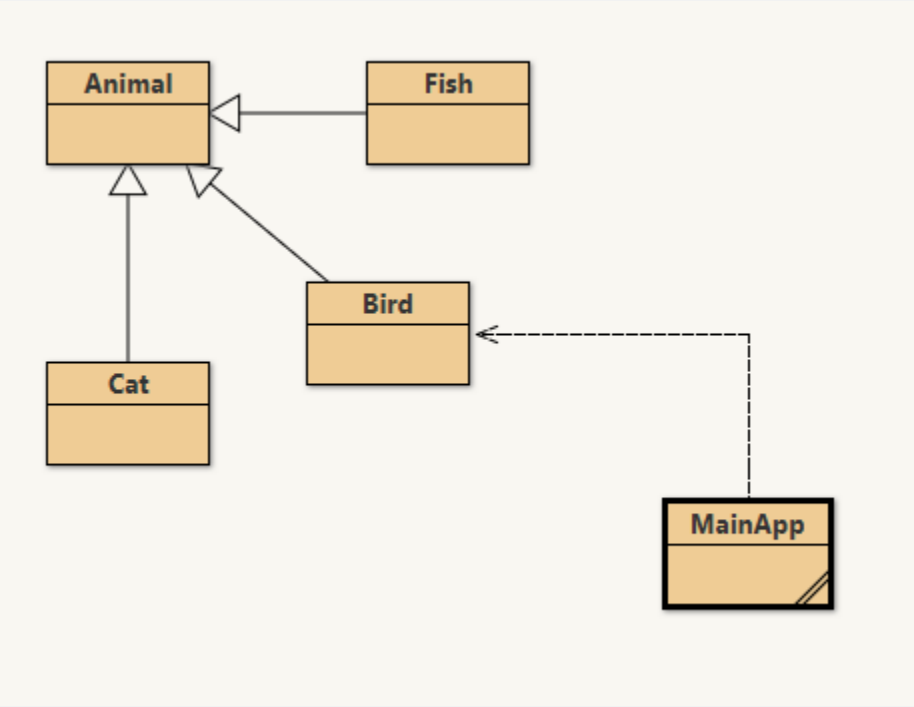
HAS-A

Any class that instantiate an object, that class HAS-A object

```
public class Animal {
    private Cat cat;
}
```

Animal HAS-A Cat

RELATIONSHIP



1. Fish IS-A animal
2. Bird IS-A animal
3. Cat IS-A animal
4. MainApp HAS-A Bird

EXERCISE #2

Create superclass and subclasses for polygons (square,pentagon,hexagon,etc)

OOP - POLYMORPHISM

- ✓ Extension of IS-A relationship between subclasses and superclass.
- ✓ Instantiate subclasses with a superclass reference.
- ✓ Only accessible to states/behaviors defined on superclass.
- ✓ If subclass override superclass states/behaviors, upon method invocation from superclass reference, it will actually invoke overridden the states/behaviors.

```
public class ShowAnimals {
    public static void main(String[] args) {
        Animal ani1 = new Cat();
        Animal ani2 = new Bird();
        Animal ani3 = new Fish();

        ani1.sleep();
        ani2.sleep();
        ani3.sleep();
    }
}
```

Instantiate Cat object with Animal as reference

Invoke Cat.sleep() since Cat object override Animal.sleep()

Invoke Animal.sleep()

OOP - POLYMORPHISM

- ✓ Original subclass instance can be assigned to superclass reference and then assign to another subclass reference.

```
public class ShowAnimals {
    public static void main(String[] args) {
        Cat cat1 = new Cat();
        Animal ani1 = cat1;
        Cat cat2 = ani1;

        Bird bird = new Bird();
        Fish fish = bird;
    }
}
```

ERROR...!!! Cannot assign superclass to subclass

ERROR...!!! Bird cannot be assigned to Fish

CASTING

- ✓ Casting is required if we only have superclass reference and we know and want to use like a subclass instance.
- ✓ Just close object name with parenthesis during assignment.

```
public class GetAnimal {
    public static Animal get(char prefix) {
        switch (prefix) {
            case 'B' : return new Bird();
            case 'C' : return new Cat();
            case 'F' : return new Fish();
        }
        return null;
    }
}
public class ShowAnimal {
    public static Animal get(char prefix) {
        Animal animal = GetAnimal.get('C');
        animal.sleep();
        Cat cat = (Cat) animal;
        cat.sleep();
    }
}
```

This object only return superclass reference

Cast to subclass object to access overridden or internal behavior

INSTANCE-OF

- ✓ We cannot tell whether superclass
- ✓ An operator to check whether an instance is referring to another instance or not.
- ✓ Usually to check whether superclass reference is an instance of which subclass.
- ✓ Keyword : instanceof

```
public class ShowAnimal {
    public static Animal get(char prefix) {
        Animal animal = GetAnimal.get('C');
        if (animal instanceof Bird) {
            System.out.println("bird");
        }
        else if (animal instanceof Cat) {
            System.out.println("cat");
        }
        else {
            System.out.println("fish");
        }
    }
}
```

Operation is between instance field and Object

OOP - ABSTRACTION

- ✓ Abstraction is a way to provide behaviors in superclass but let the subclasses do the implementation without overriding parent behaviors.
- ✓ Keyword : abstract (class and method)

Abstract class

```
public abstract class Animal {
    ...

    public abstract void eat();
}
```

Abstract method ends with semicolon, no braces

IMPLENTATION

- ✓ All subclasses need to implement the abstract method

```
public class Cat extends Animal {
    ...

    @Override
    public void eat() {
        System.out.println("Fish");
    }
}
public class Bird extends Animal {
    ...

    @Override
    public void eat() {
        System.out.println("Insect");
    }
}
public class Fish extends Animal {
    ...

    @Override
    public void eat() {
        System.out.println("Algae");
    }
}
```

Implementation of abstract method from superclass

INVOKE

- ✓ Casting is not required to invoke abstract method from superclass reference.

```
public class ShowAnimalEat {
    public static Animal get(char prefix) {
        Animal animal = GetAnimal.get('C');
        animal.eat();

        Bird bird = new Bird();
        bird.eat();
    }
}
```

EXERCISE #3

Create superclass and subclasses for family members (father, mother, brother, sister) with name, age and abstraction to show their occupation.

OOP - INTERFACE

- ✓ Interface is another blueprint as a basic structure to be implemented into subclasses (implementor)
- ✓ Usually if all methods must be implements by subclass
- ✓ Only final/constant fields allowed
- ✓ All methods ends with semicolon, like abstract method, but without abstract keyword

"interface" keyword instead of class

```
public interface IAnimal {
    public static final String TYPE = "Animal";

    public void sleep();

    public void eat();
}
```

Methods to be implemented by implementors

IMPLEMENTATION


```
public class Cat implements IAnimal {
    @Override
    public void sleep() {
        System.out.println("box");
    }

    @Override
    public void eat() {
        System.out.println("fish");
    }
}
```

"implements" keyword to implement an interface

```
public class Bird implements IAnimal {
    @Override
    public void sleep() {
        System.out.println("nest");
    }

    @Override
    public void eat() {
        System.out.println("insect");
    }
}
```

Methods implementations based on interface

IMPLEMENTATION (2/2)

```
public class Fish implements IAnimal {
    @Override
    public void sleep() {
        System.out.println("rock");
    }

    @Override
    public void eat() {
        System.out.println("algae");
    }
}
```

INVOKE

- ✓ Casting is not required to invoke abstract method from superclass reference.

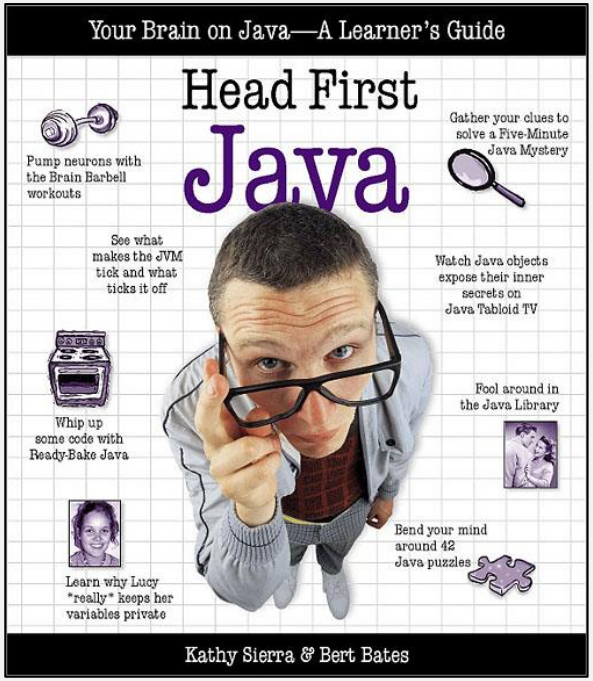
Interface as a reference instead of object

```
public class ShowAnimals {
    public static Animal get(char prefix) {
        IAnimal animal = GetAnimal.get('C');
        animal.eat();
        animal.sleep();

        Bird bird = new Bird();
        bird.eat();
    }
}
```

EXERCISE #4

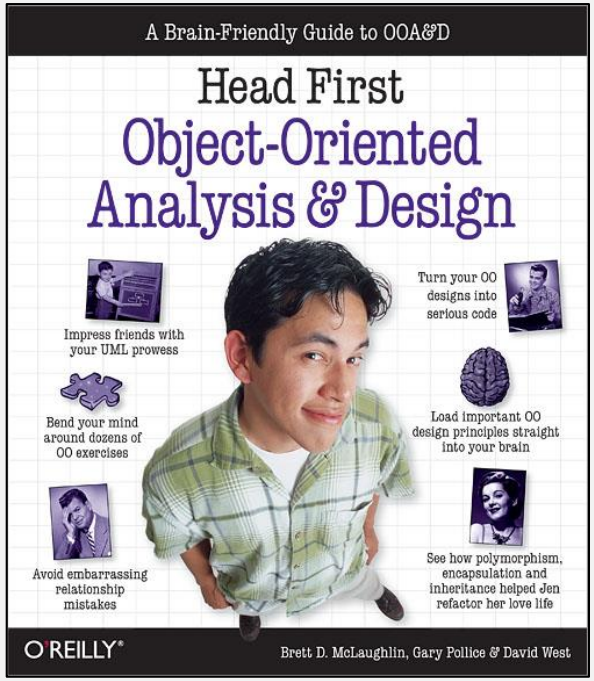
Create interface and implementation for 3 shapes (square, star, circle) on number of sides, corner type and calculate area.

FURTHER READING



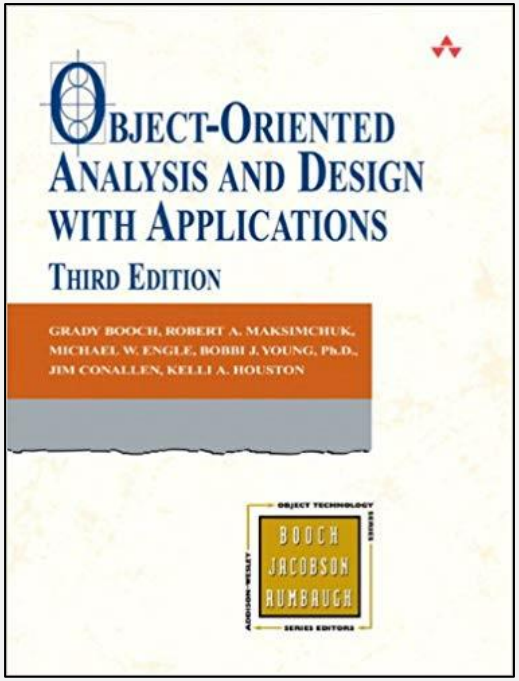
Head First Java (2nd Edition)

by Kathy Sierra (Author), Bert Bates (Author)



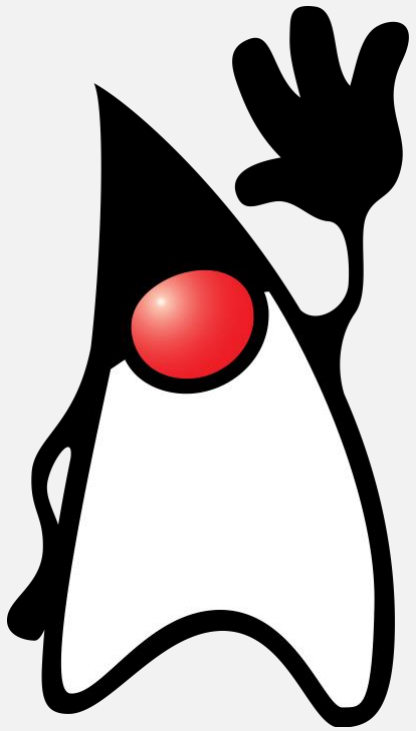
Head First Object-Oriented Analysis and Design

Brett D. McLaughlin, Gary Pollice, Dave West



Object-Oriented Analysis and Design with Applications (3rd Edition)

Grady Booch, Robert A. Maksimchuk, Michael W. Engle



THAT'S ALL FOR TODAY
SEE YOU IN THE NEXT CLASS