Object-Oriented Software Engineering

Chapter 2:
Review of Object Orientation

2.1 What is Object Orientation?

Procedural Oriented Programming:

- Each steps is executed in a systematic manner so that the computer can understand what to do (step by step).

- *C called procedural orientation language because large program is divided into modules.*
  - —Groups together the pieces of data that describe some entity
  - —Helps reduce the system's complexity.

Object oriented Programming:

- Organizing your code by creating objects, and then you can give those objects properties.

Object Oriented

An approach to the solution of problems in which all computations are performed in the context of objects.

- The objects are instances of classes, which:
  - —are data abstractions
  - —contain procedural abstractions that operate on the objects
- A running program can be seen as a collection of objects collaborating to perform a given task

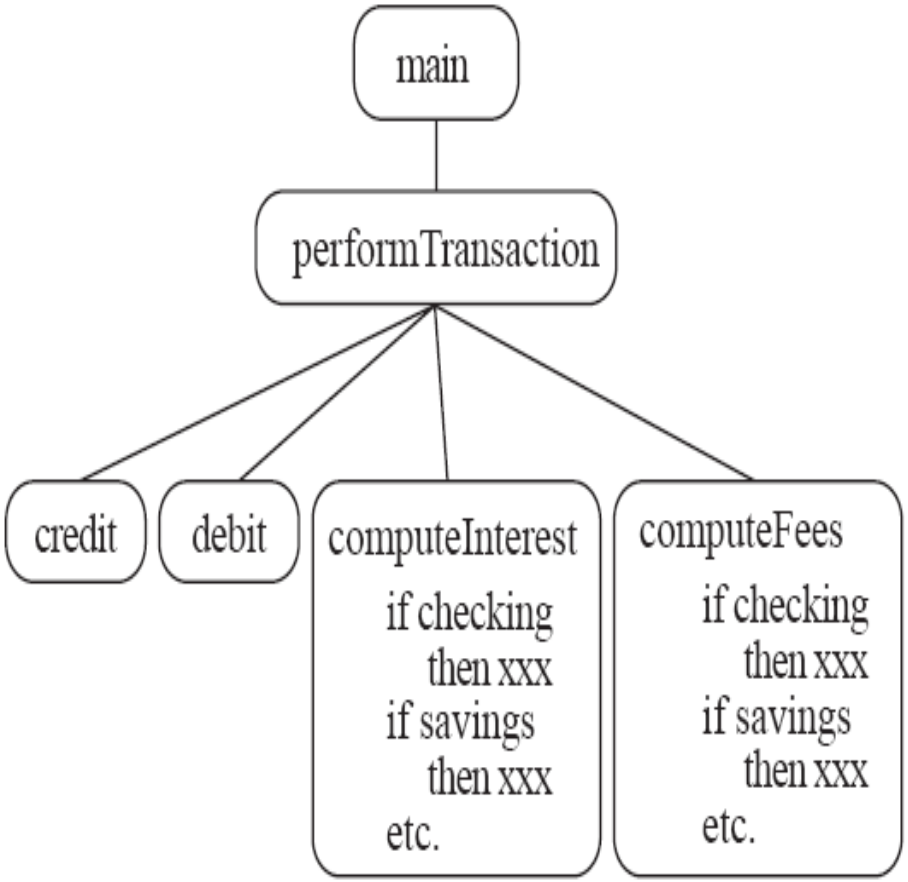
Assignments 1

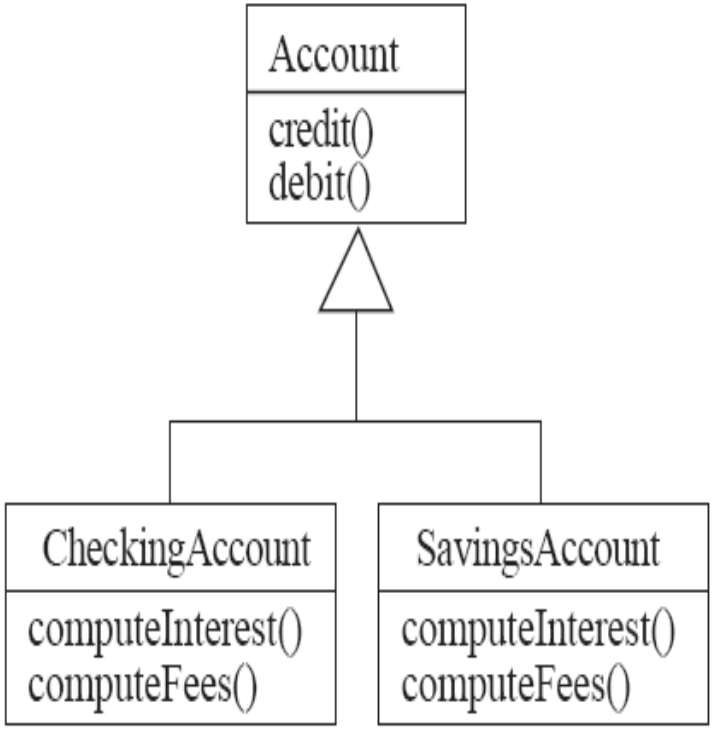
What are a differences between procedural and object oriented programming?

Parameter :

- Definition
- Approach
- Access modifiers
- Complexity
- Inheritance

A View of the Two paradigms





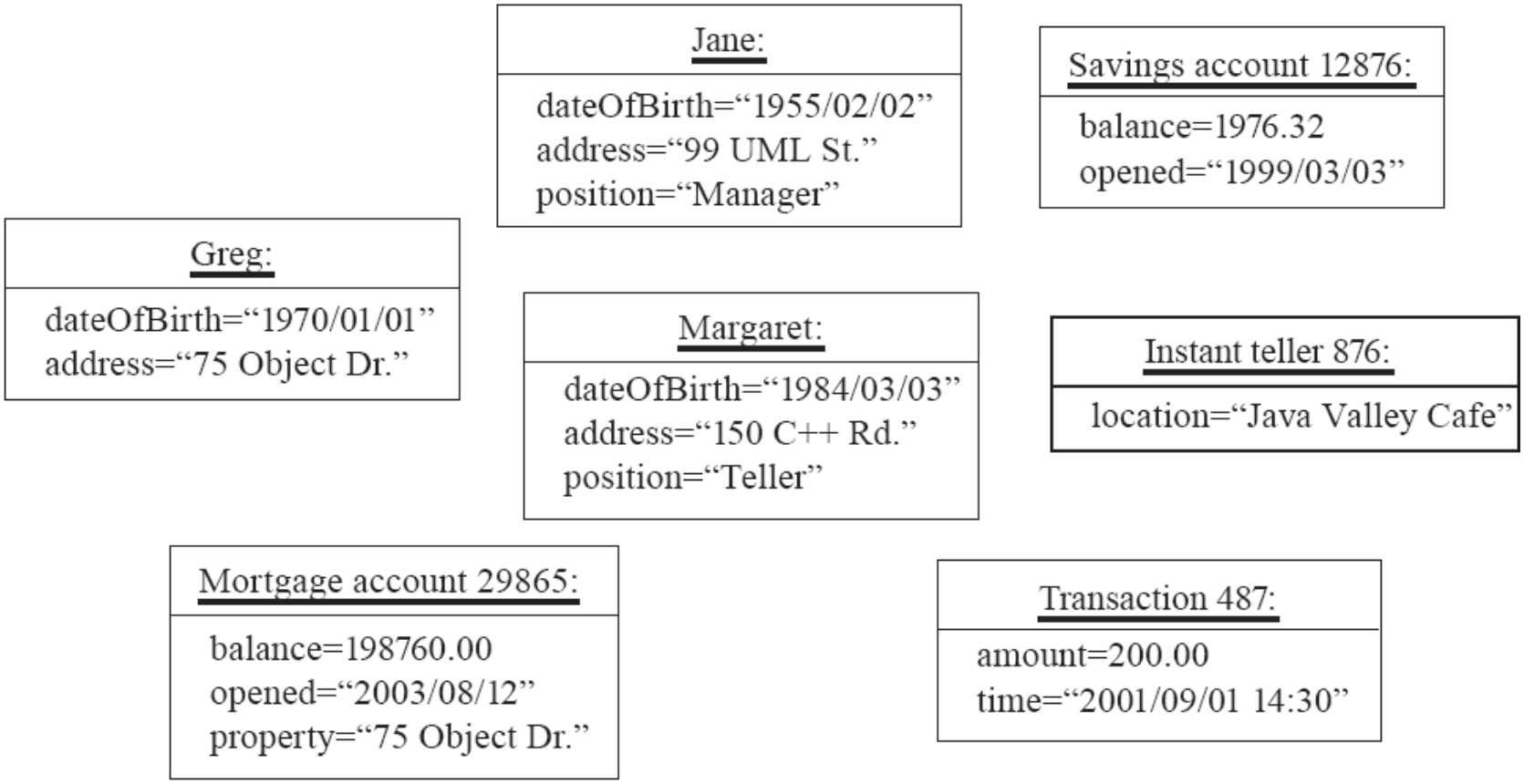
See in Umple

2.2 Classes and Objects

Object

- A chunk of structured data in a running software system
- Has *properties*
  - —Represent its state
- Has *behaviour*
  - —How it acts and reacts
  - —May simulate the behaviour of an object in the real world

Objects



Classes

A class:

- A unit of abstraction in an object oriented (OO) program
- Represents similar objects
  - —Its *instances*
- A kind of software module
  - —Describes its instances' structure (properties)
  - —Contains *methods* to implement their behaviour

Naming classes

- Use *capital* letters
  - —E.g. `BankAccount` not `bankAccount`
- Use *singular* nouns
- Use the right level of generality
  - —E.g. `Municipality`, not `City`
- Make sure the name has only *one* meaning
  - —E.g. 'bus' has several meanings

2.3 Instance Variables

Variables defined inside a class and are used to store values in an object

- Also called *fields* or *member variables*

- Attributes
  - —Simple data
  - —E.g. `name`, `dateOfBirth`

- Associations
  - —Relationships to other important classes
  - —E.g. `supervisor`, `coursesTaken`
  - —More on these in Chapter 5

Variables vs. Objects

A variable

- *Refers* to an object
- May refer to different objects at different points in time

An object can be referred to by several different variables at the same time

***Type* of a variable**

- Determines what classes of objects it may contain

Class variables

A *class variable's* value is *shared* by all instances of a class.

- Also called a *static* variable
- If one instance sets the value of a class variable, then all the other instances see the same changed value.
- Class variables are useful for:
  - —Default or 'constant' values (e.g. PI)
  - —Lookup tables and similar structures

Caution: *do not over-use class variables*

2.4 Methods, Operations and Polymorphism

Operation

- A higher-level procedural abstraction that specifies a type of behaviour

- Independent of any code which implements that behaviour

  —E.g. calculating area (in general)

Methods, Operations and Polymorphism

Method

- A procedural abstraction used to implement the behaviour of a class

- Several different classes can have methods with the same name
  - —They implement the same abstract operation in ways suitable to each class
  - —E.g. calculating area in a rectangle is done differently from in a circle

Polymorphism

A property of object oriented software by which an *abstract operation may be performed in different ways* in different classes.

- Requires that there be *multiple methods of the same name*
- The choice of which one to execute depends on the object that is in a variable
- Reduces the need for programmers to code many `if-else` or `switch` statements

2.5 Organizing Classes into Inheritance Hierarchies

Superclasses

- Contain features common to a set of subclasses

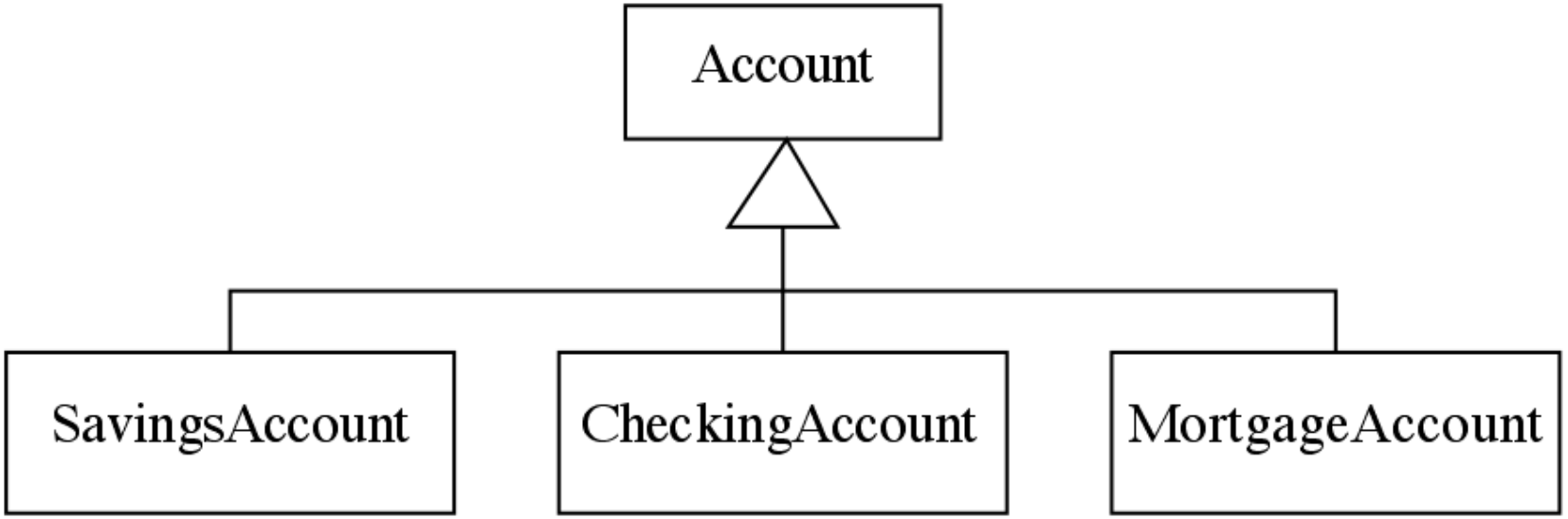
Inheritance hierarchies

- Show the relationships among superclasses and subclasses
- A triangle shows a *generalization* 

Inheritance

- The *implicit* possession by all subclasses of features defined in its superclasses

An Example Inheritance Hierarchy

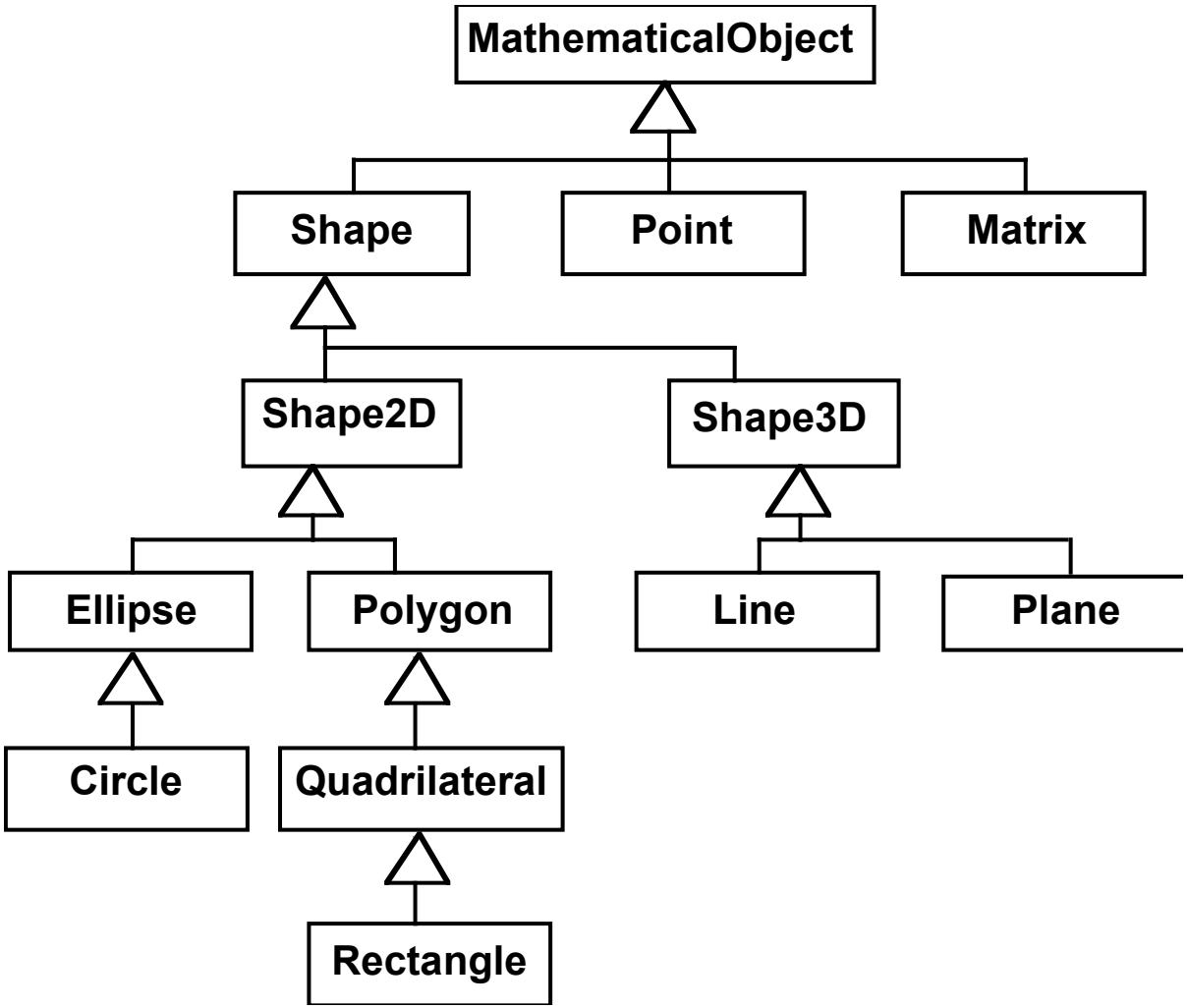


See in Umple

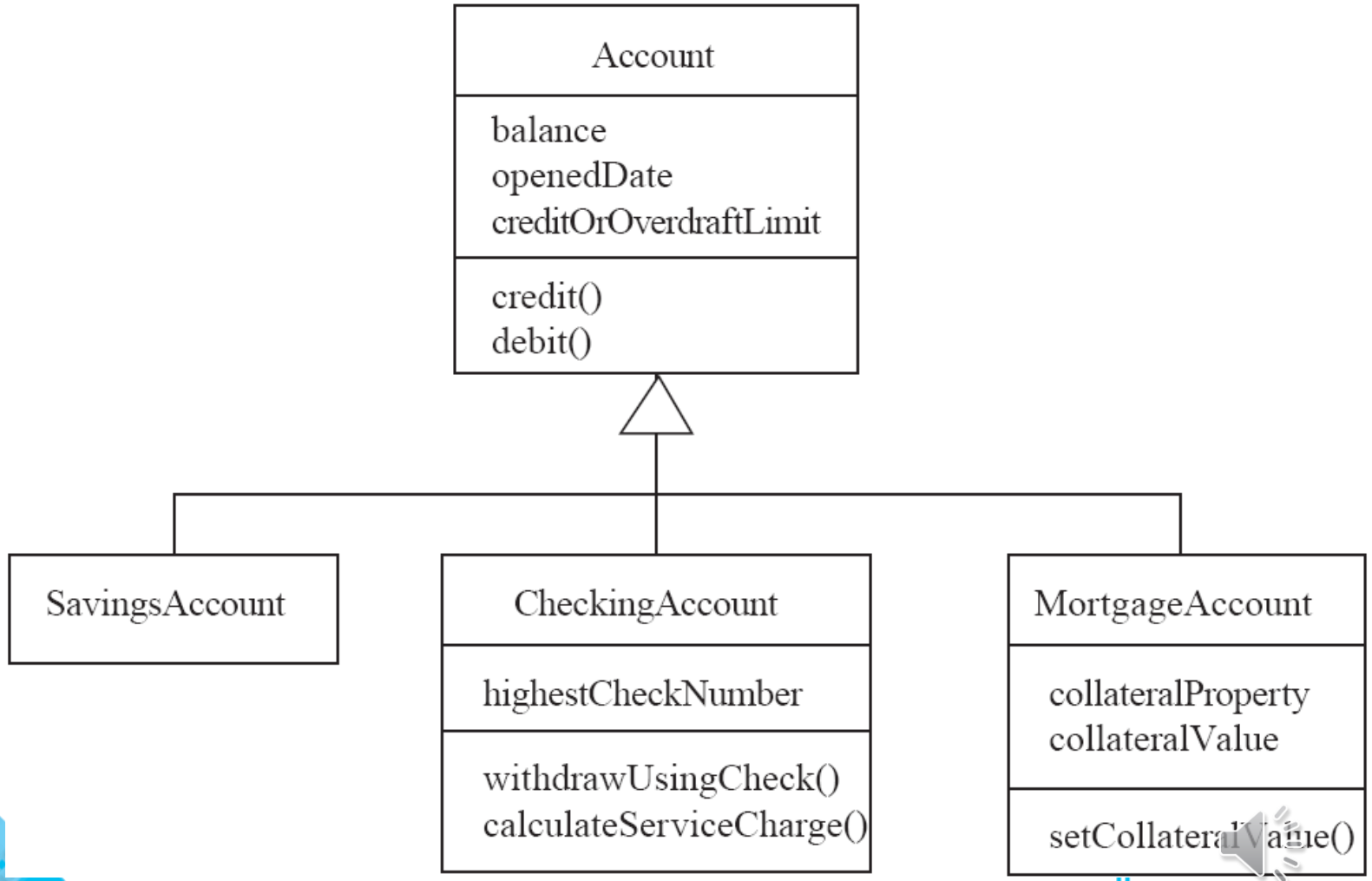
Inheritance

- The *implicit* possession by all subclasses of features defined in its superclasses

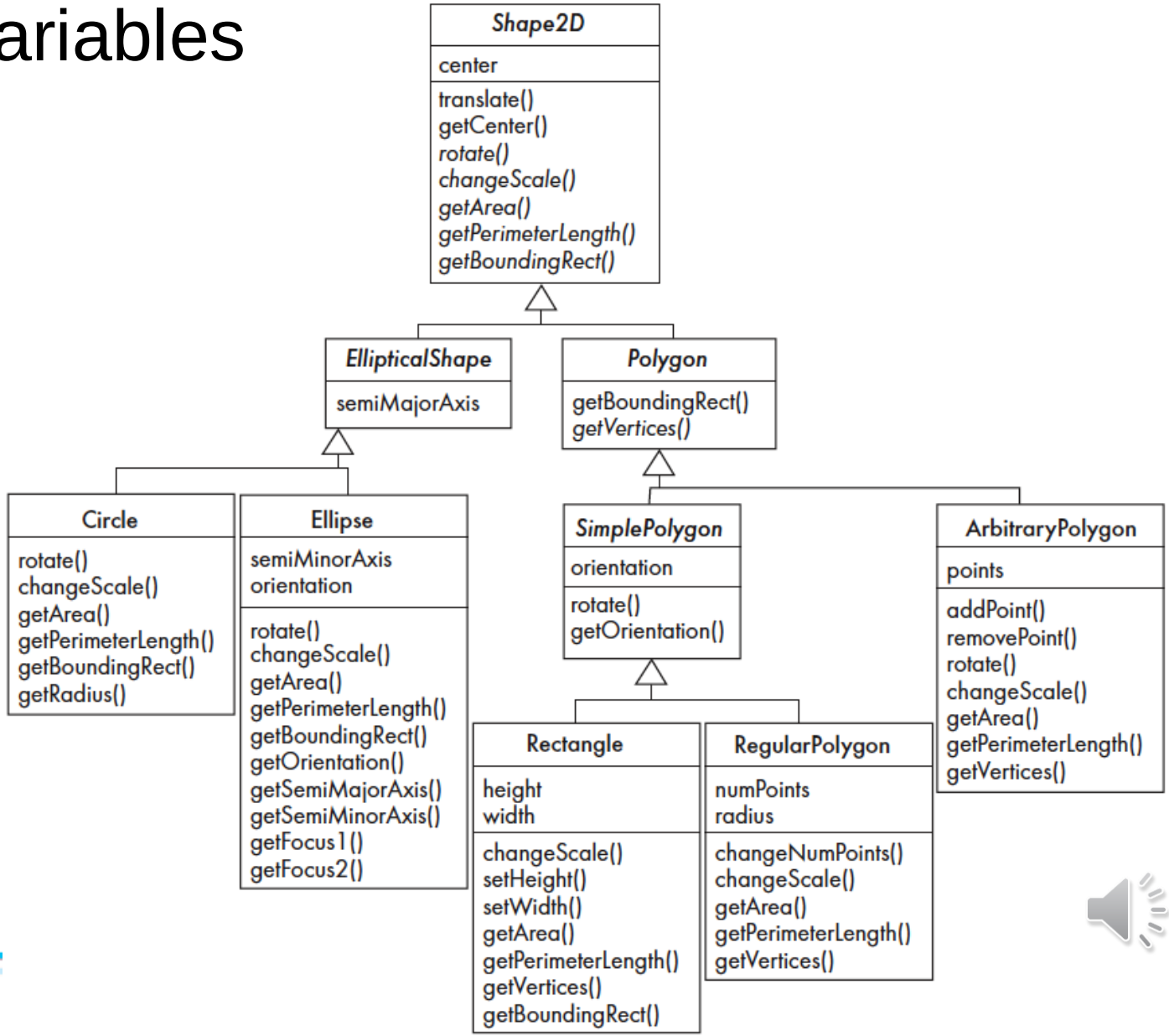
A possible inheritance hierarchy of mathematical objects



Make Sure all Inherited Features Make Sense in Subclasses



2.6 Inheritance, Polymorphism and Variables



Abstract Classes and Methods

An operation should be declared to exist at the highest class in the hierarchy where it makes sense

- The *operation* may be *abstract* (lacking implementation) at that level
- If so, the *class* also must be *abstract*
  - —No instances can be created
- If a superclass has an abstract operation then its subclasses at some level must have a concrete method for the operation
  - —Leaf classes must have or inherit concrete methods for all operations
  - —Leaf classes must be concrete

Overriding

A method would be inherited, but a subclass contains a new version instead

- For restriction
  - —E.g. `scale(x,y)` would not work in `Circle`
- For extension
  - —E.g. `SavingsAccount` might charge an extra fee following every debit
- For optimization
  - —E.g. The `getPerimeterLength` method in `Circle` is much simpler than the one in `Ellipse`

2.7 Concepts that Define Object Orientation

The following are necessary for a system or language to be OO

- Identity
  - —Each object is *distinct* from each other object, and *can be referred to*
  - —Two objects are distinct *even if they have the same data*
- Classes
  - —The code is organized using classes, each of which describes a set of objects
- Inheritance
  - —The mechanism where features in a hierarchy inherit from superclasses to subclasses
- Polymorphism
  - —The mechanism by which several methods can have the same name and implement the same abstract operation.

Other Key Concepts

Abstraction

- Object -> something in the world
- Class -> objects
- Superclass -> subclasses
- Operation -> methods
- Attributes and associations -> instance variables

Modularity

- Code can be constructed entirely of classes

Encapsulation

- Details can be hidden in classes
- This gives rise to *information hiding*:
  —Programmers do not need to know all the details of a class

Access control

Applies to methods and variables

- `public`
  - —Any class can access
- `protected`
  - —Only code in the package, or subclasses can access
- (blank)
  - —Only code in the package can access
- `private`
  - —Only code written in the class can access
  - —Inheritance still occurs!

Programming Style Guidelines

Remember that programs are for people to read

- Always choose the simpler alternative
- Reject clever code that is hard to understand
- Shorter code is not necessarily better

Choose good names

- Make them highly descriptive
- Do not worry about using long names

Programming style …

Comment extensively

- Comment whatever is non-obvious
- Do not comment the obvious
- Comments should be 25-50% of the code

Organize class elements consistently

- Variables, constructors, public methods then private methods

Be consistent regarding layout of code

Programming style …

Avoid duplication of code

- Do not ‘clone’ if possible
  - —Create a new method and call it
  - —Cloning results in two copies that may both have bugs
    - When one copy of the bug is fixed, the other may be forgotten

2.10 Difficulties and Risks in Object-Oriented Programming

Language evolution and deprecated features:

- Java is evolving, so some features are 'deprecated' at every release
- But the same thing is true of most other languages

Efficiency can be a concern in some object oriented systems

- Java can be less efficient than other languages
  - —VM-based
  - —Dynamic binding