BINUS UNIVERSITY INTERNATIONAL

People
Innovation
Excellence

The Object Oriented Programming Paradigm

Concept of Programming Paradigms

- How we think/view/implement a program
- Specific programming languages are designed to support specific programming paradigm

Procedural Programming Paradigm

- Program is an explicit series of instructions for the computer to execute.
- Step – by – step instructions are specified to achieve the task on hand
- Structured programming constructs include sequence, selection and loop

Structured Programming Constructs

- Sequential
  - Instructions are executed in sequence
- Selection
  - Instructions are executed based on a certain condition
- Loop (iteration)
  - Repetition of a single or group of statements based on a condition

Logic Programming Paradigm

- Program consists of declaratives (facts and rules)
- Programmer builds knowledge base of facts and rules about a certain domain of interest – this knowledge base constitutes the program
- Interaction with the program takes place by posing queries to an inference engine (also called query interpreter)
- The inference engine can access the knowledge base and has its own rules of deductive reasoning
- The programmer need not instruct the inference engine when a rule should be applied

Functional Programming Paradigm

- A functional programming language (FPL) views every task in terms of functions
- A function in FPL is like a mathematical function, taking one or more arguments and transforming their values into a single value according to some specified value
  - E.g. With a function f(x) = 2x, f(3) = 6, f(6) = 12
- Primitive functions or primitives are functions that are defined as part of the language
- Programmer can define and name his own functions

Functional Programming Languages

- LISP (LISt Processing) programming language
- FP (for Functional Programming) language
- SCHEME

An Example Using SCHEME

- (define (double x)
  (* 2 x))
- (define (square x)
  (* x x))
- (define (polynomial x)
  (double (square x)))

LISP Primitive Functions

- Arguments to functions are frequently lists of things
- A list can be an empty list called “nil”
- List Primitive functions:
  - list e.g. (list 3 4 5)
    - Creates a list
  - car e.g. (car (3 4 5))
    - Accepts the list and removes first character then displays it
  - cdr e.g. (cdr (3 4 5))
    - Accepts the list and removes the first character then displays remaining characters
  - null?
    - Returns Y/N, T/F

A Full Example

- (define (adder input-list)

  (cond ((null ? input – list) 0)

  (else (+ (car input – list)

  adder (cdr input – list))))))

- What is the answer to (adder (list 3 4 5))?

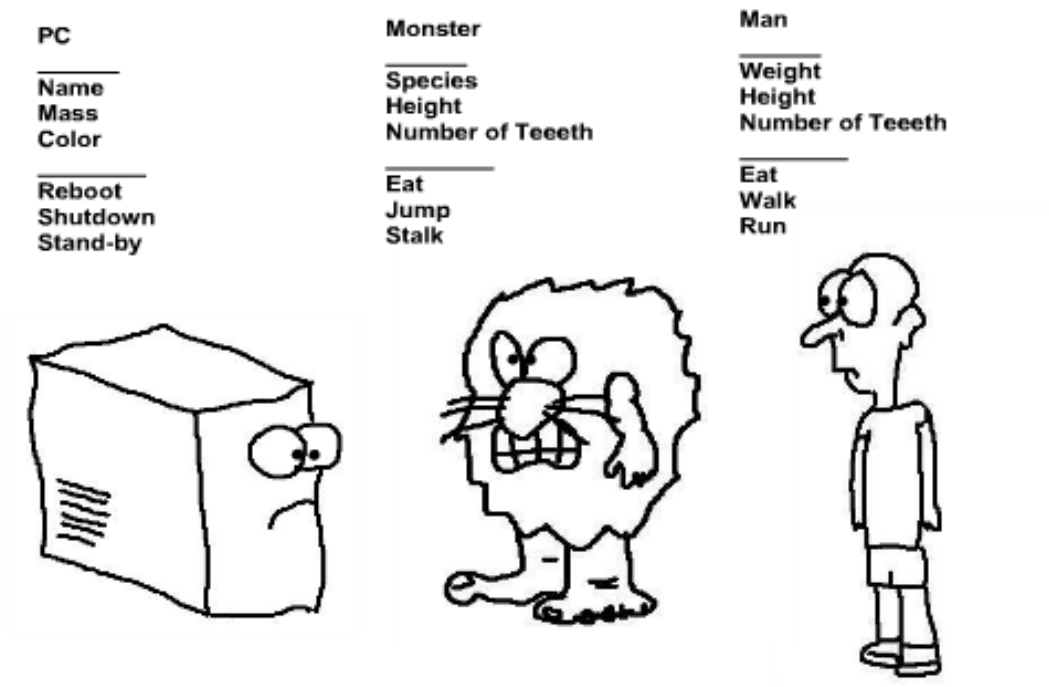
Understanding the Object Paradigm

- Object-oriented programming (OOP) is a programming paradigm that seeks to solve problems in terms of **objects** and their interactions.
- Style of programming based on objects
- OOP helps you solve difficult problems by providing techniques to model real-life conditions in code.

What is an object?

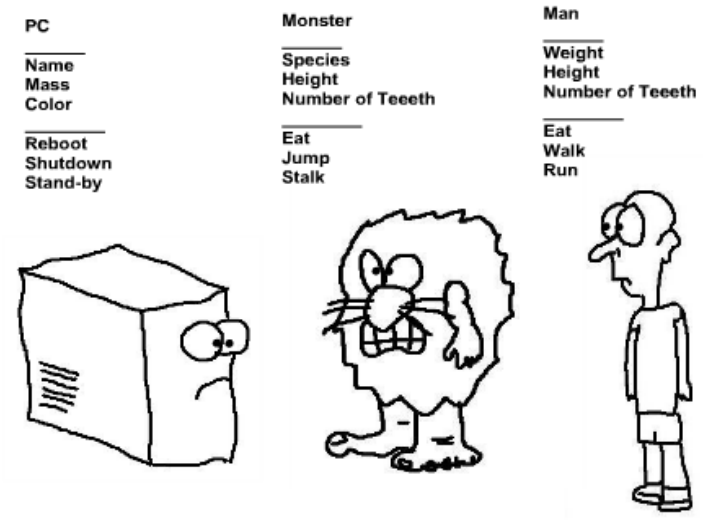
- Anything that represents any real-world thing or entity
- Has properties and methods
- Instance of a class

An object is "instantiated" from a class



What is an object?

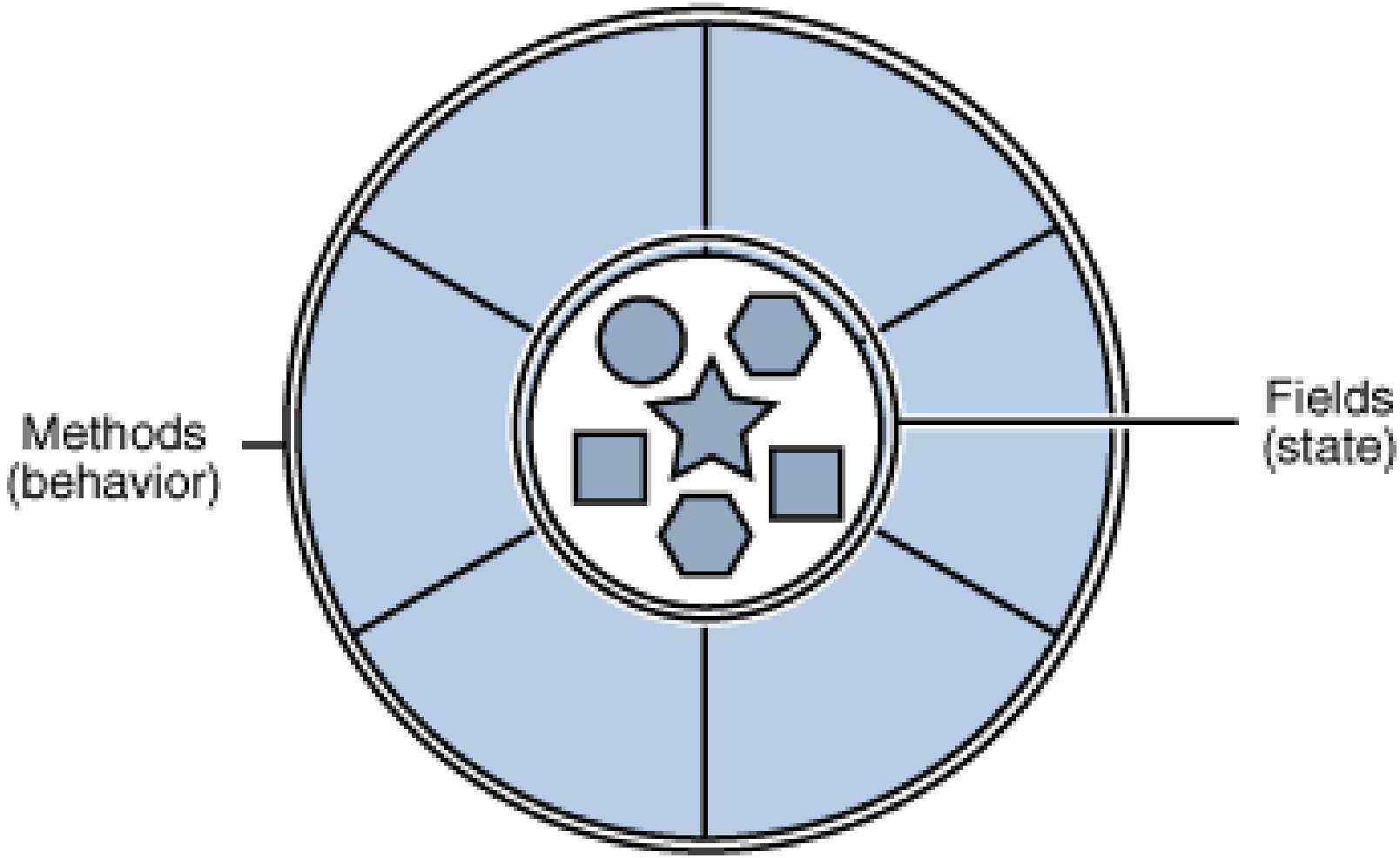
- An entity with attributes and behavior encapsulated.
- Attributes and behavior are called member variables and methods respectively in Python.
- Attributes have values that you can “set” and “get”
- Methods are operations that an object can do



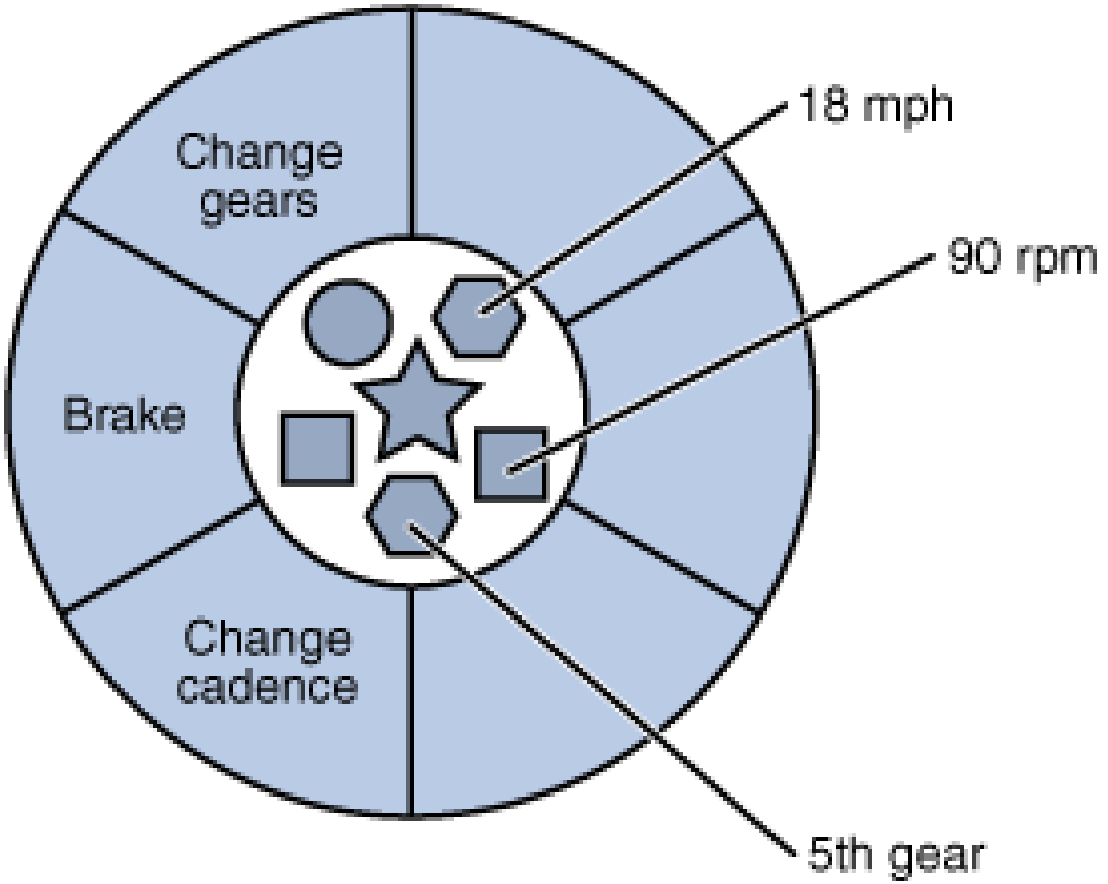
Exercise…

Take a minute right now to observe the real-world objects that are in your immediate area. For each object that you see, ask yourself two questions: "What possible states can this object be in?" and "What possible behavior can this object perform?".

A Software Object

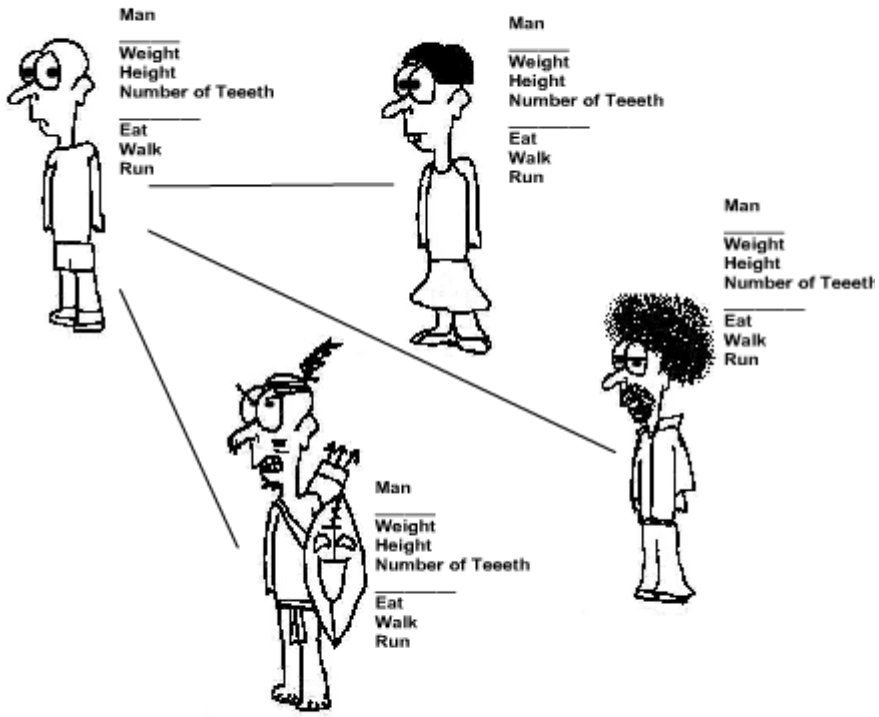


A bicycle modeled as a Software Object



What is a class?

- Template for creating objects
- Template to specify the attributes and behavior (methods) of a type of object
- Prototype
- Used to instantiate objects
- Objects created from a class will exhibit the attributes and behavior defined in the class



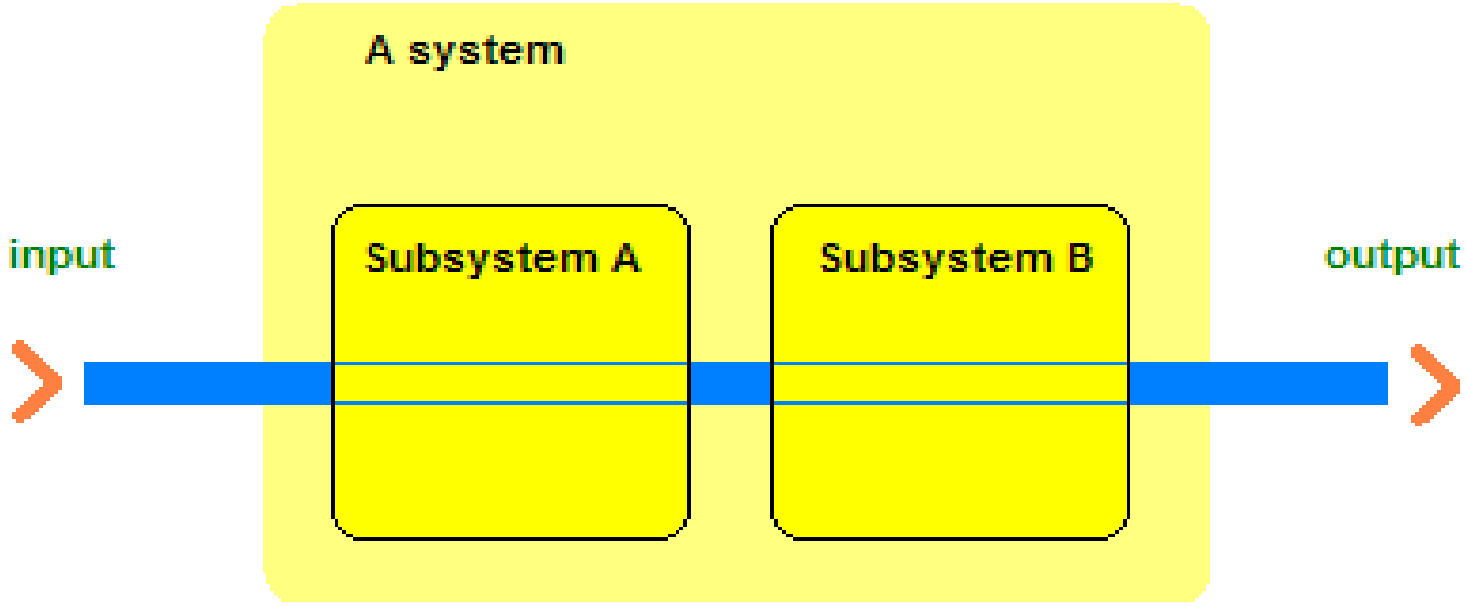
Core Principles on OOP

- Modularity
- Encapsulation
- Inheritance
- Polymorphism

Concept of Modularity

- The source code for an object can be written and maintained independently of the source code for other objects. Once created, an object can be easily passed around inside the system.

Concept of Modularity



Concept of Encapsulation and Interfaces

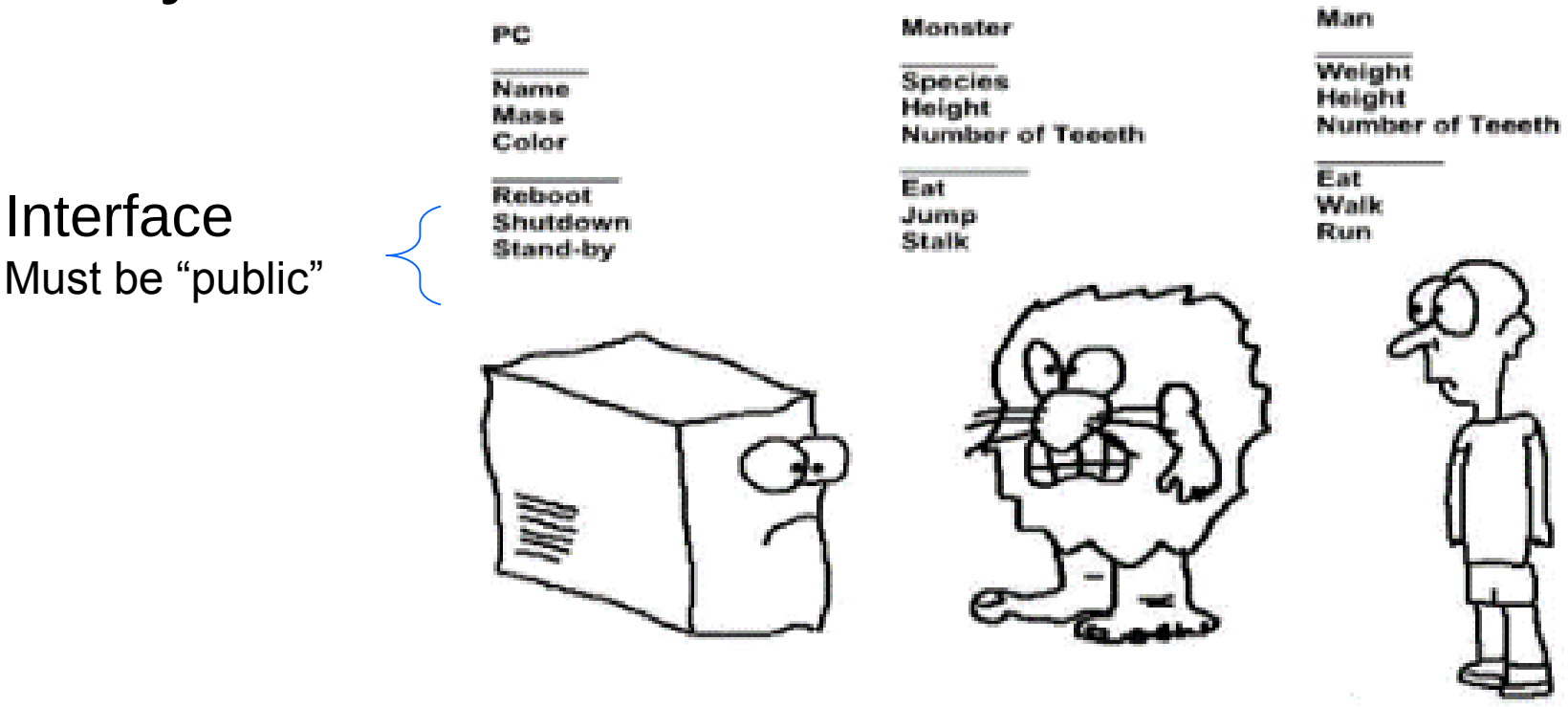
- It says that everything except the interface of an object should be hidden, so that the actual implementation can easily be changed independently. As a bonus, this also improves program security.

Concept of Encapsulation and Interfaces

- Encapsulation refers to the packaging of attributes (data) and behavior (methods) into a single entity.
- Attributes are private and are accessed by clients through public accessor and mutator methods
- This results in data hiding
- The public accessor and mutator methods form the public interface of an object.

Encapsulation

- All properties, methods and procedures are wrapped and integrated around an object



Class interface

- Group of methods that allow interaction to happen to an object;
- Adheres to the idea of information hiding
- The object protects its properties by exposing or hiding appropriate methods
- Depends on access specifiers

Advantages of Encapsulation

- State of an object (data) is not exposed to direct manipulation by a client
- Any changes to the state of an object can be validated
- Internal implementation of data can be changed without affecting clients
- Implementation of methods can be changed without affecting clients so long as the public interface remains the same

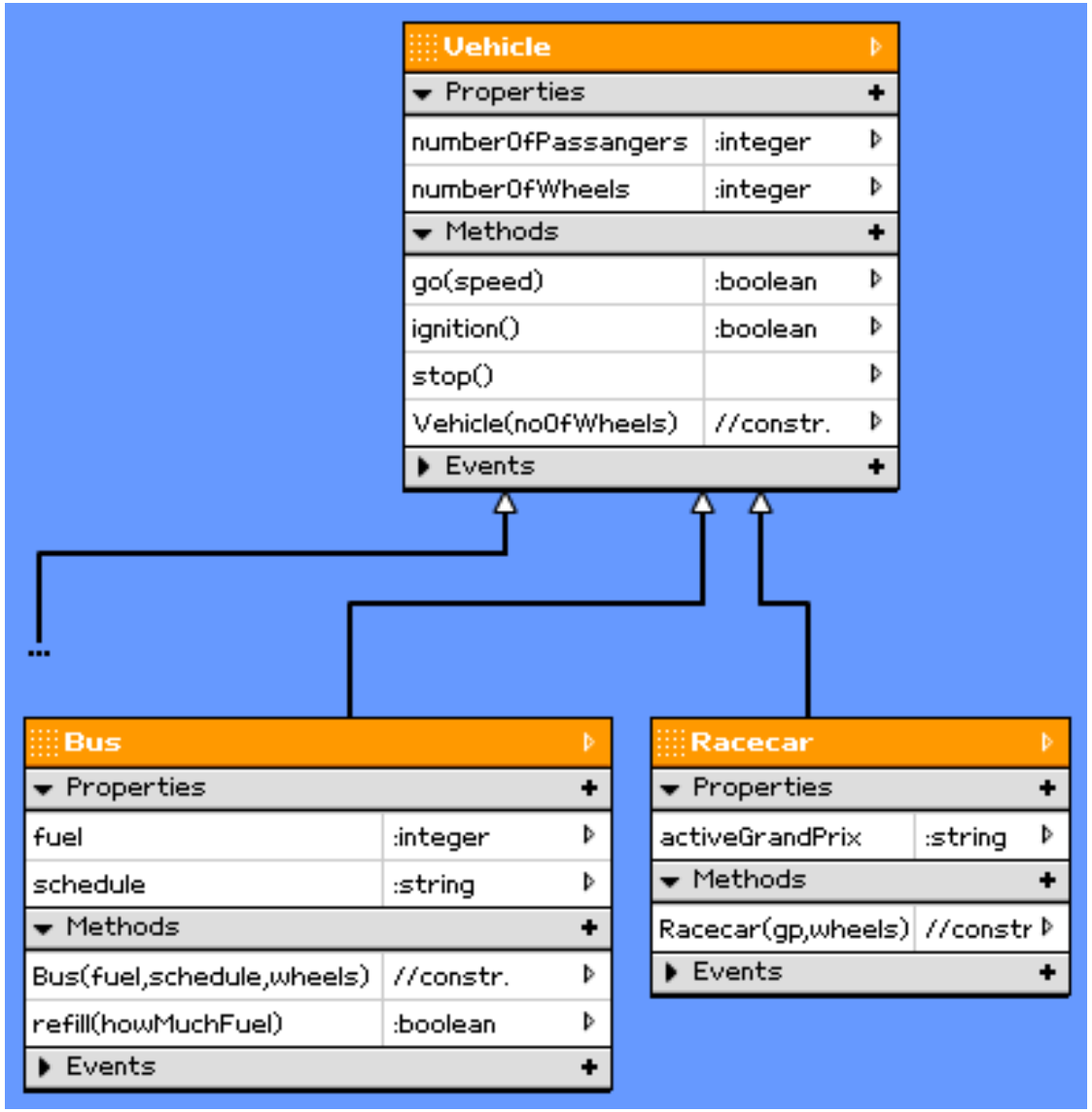
Bundling code into individual software objects provides a number of benefits, including:

- Information-hiding: By interacting only with an object's methods, the details of its internal implementation remain hidden from the outside world.
- Code re-use: If an object already exists (perhaps written by another software developer), you can use that object in your program. This allows specialists to implement/test/debug complex, task-specific objects, which you can then trust to run in your own code.
- 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. This is analogous to fixing mechanical problems in the real world. If a bolt breaks, you replace *it*, not the entire machine.

What is Inheritance?

- Inheritance allows a developer to define an object to exhibit characteristics of another object
- The specialized class is called the subclass while the generalized one is called the super class
- Subclass can have its own unique characteristics
- Greater reusability and productivity is achieved

Inheritance



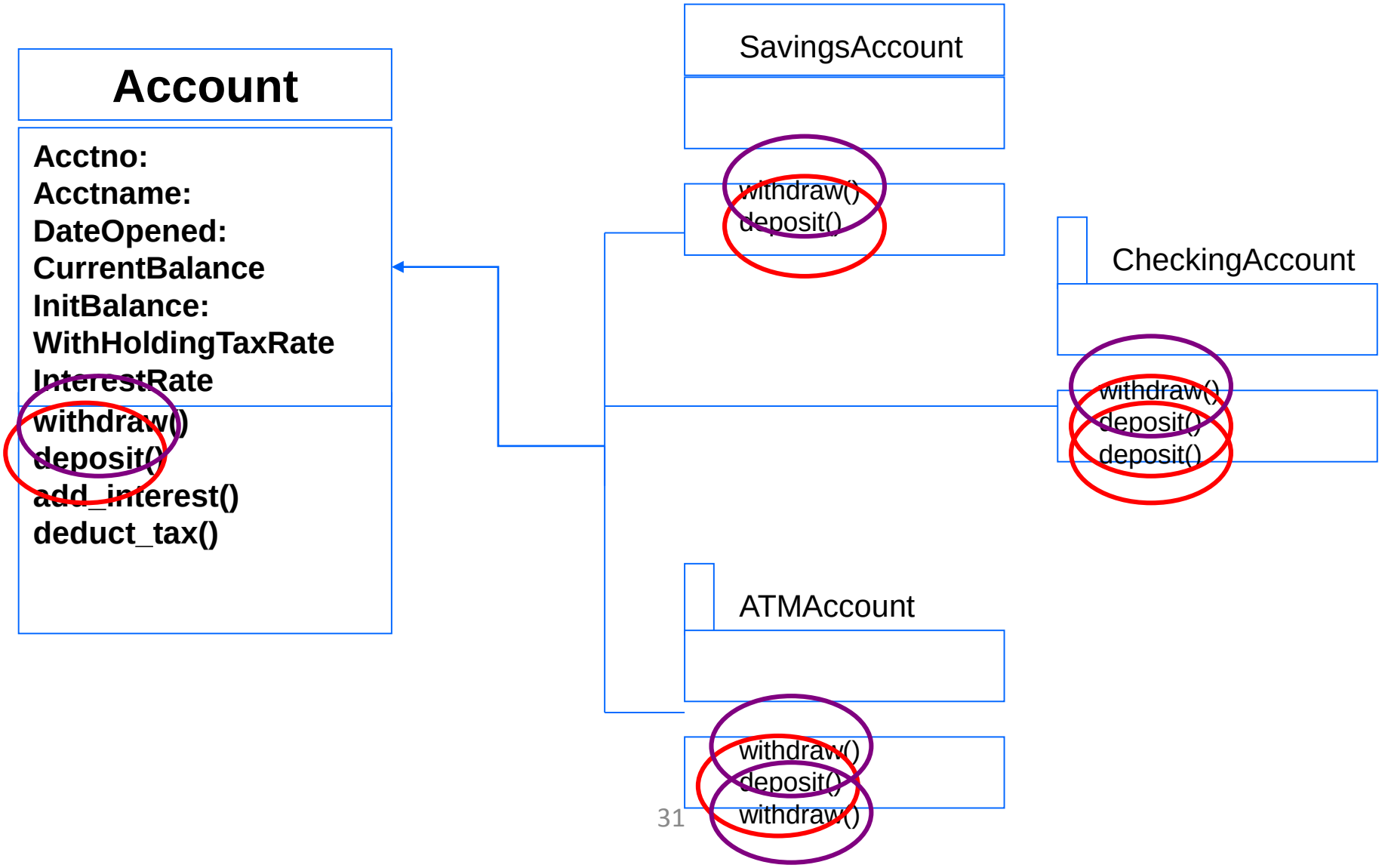
“Is – a” Relationship

- The subclass and the super class form an “is-a” relationship
- Since the subclass is a specialized “version” of the super class, the subclass object can be viewed “as an” object of the same type as the super class.

Polymorphism and Dynamic Binding

- Polymorphism is the ability to enable different behavior by sending the same message to different objects
- Polymorphism is made possible through both inheritance and dynamic or late binding
- During runtime, the nature of the target object is determined and the appropriate method is invoked. This is known as dynamic binding or late binding.

Polymorphism – “many forms”

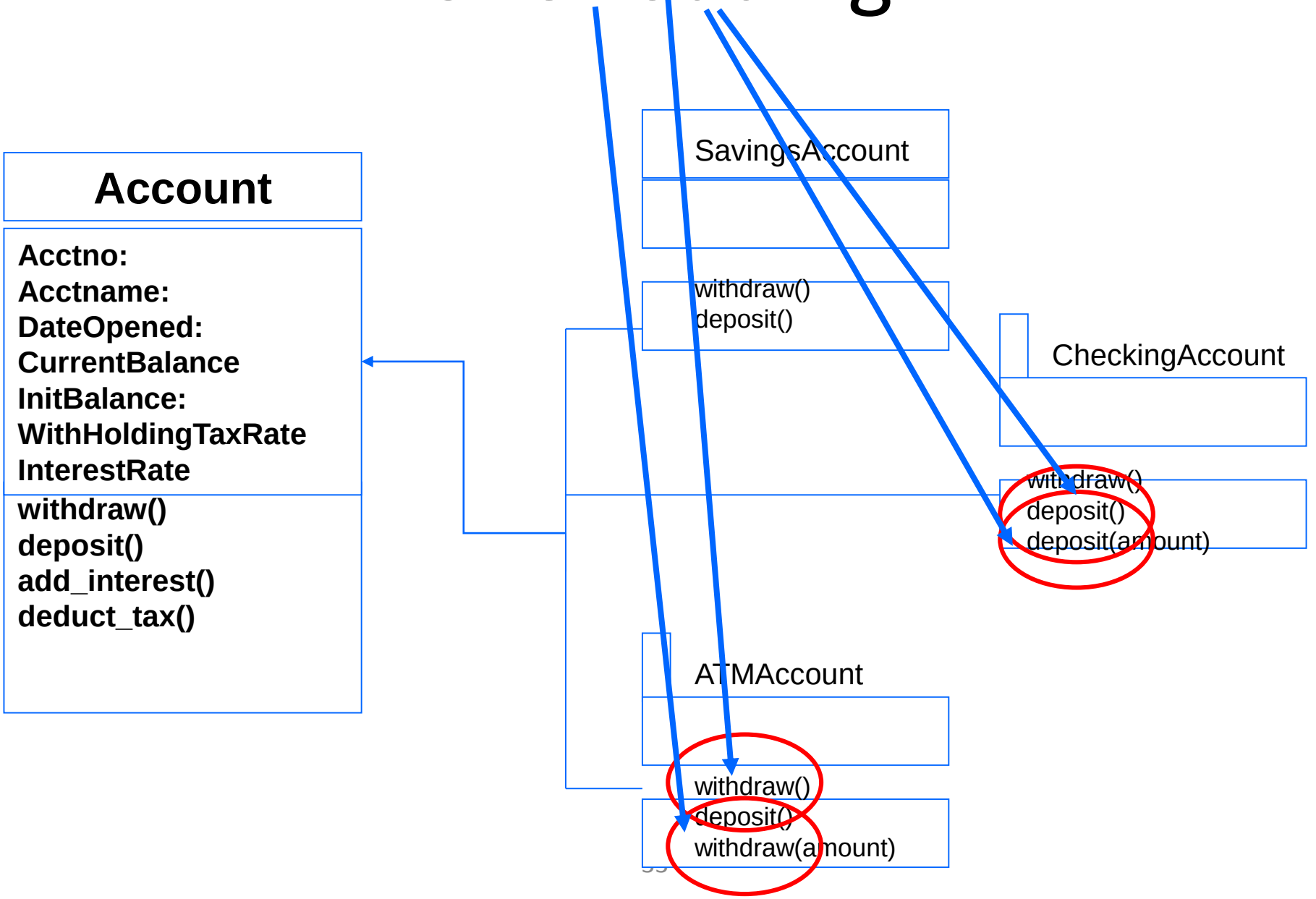


Factors that contribute to Polymorphism

- Overloading – a method has many implementations within a class; a method is found to have more than one work or way of doing its specific objective- -it is “overloaded”

- a method’s purpose depends on what data you give to it—normally defined as the function signature

Overloading

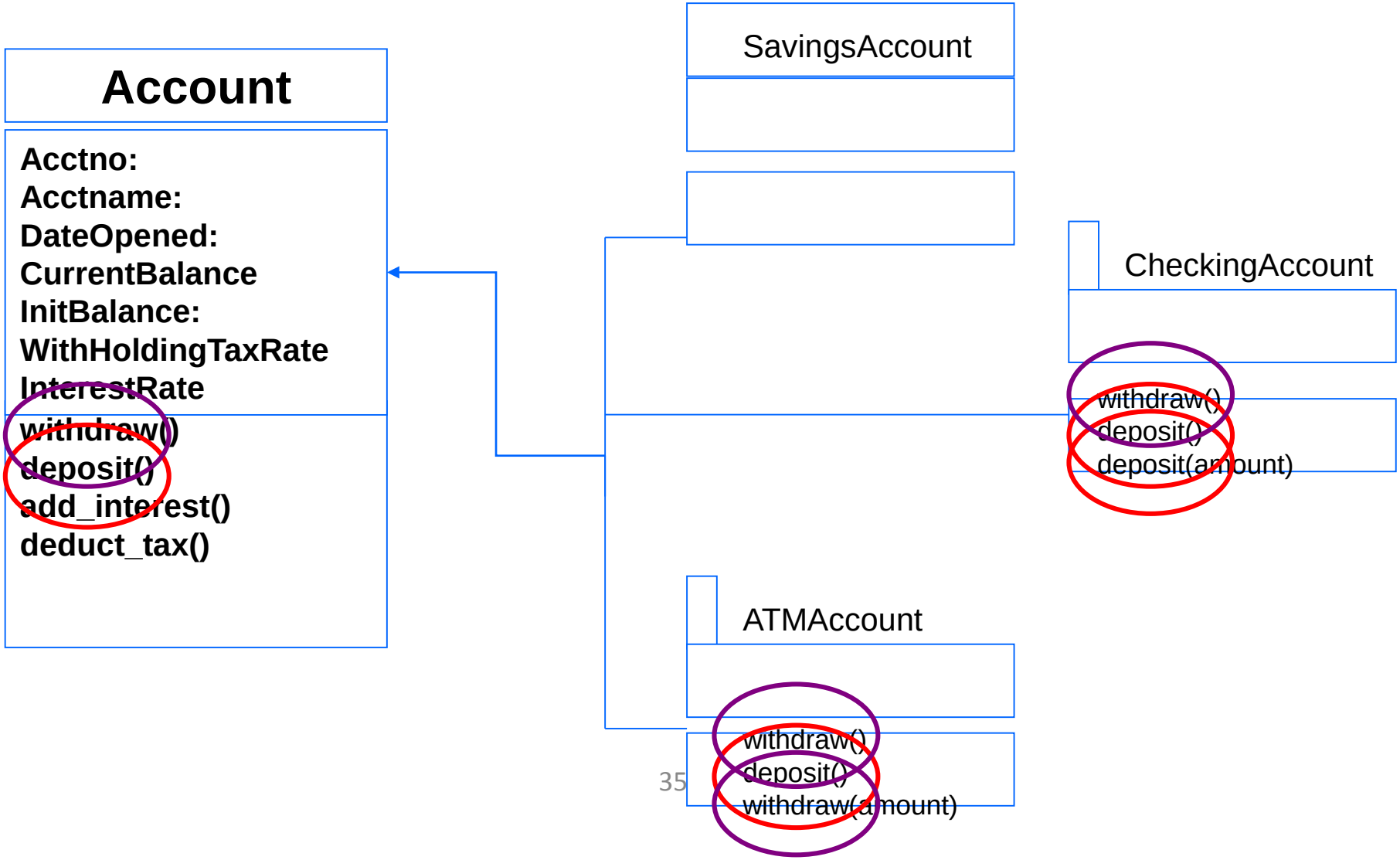


Factors that contribute to Polymorphism (cont)

- Overriding – a method's implementation in a superclass is modified by a method with the same name in a subclass ; the superclass's method is "overridden" by its subclass's method of the same name

-any object instantiated from the subclass uses the method in that subclass, not the method on the superclass

Overriding


Account
Acctno:
Acctname:
DateOpened:
CurrentBalance
InitBalance:
WithHoldingTaxRate
InterestRate
withdraw()
deposit()
add_interest()
deduct_tax()
SavingsAccount
CheckingAccount
withdraw()
deposit()
deposit(amount)
ATMAccount
withdraw()
deposit()
withdraw(amount)

Object Centered Design

1. State precisely what program should do.
2. Identify ***objects.***
   - some can be represented with primitive types
   - some require the definition of a new class
3. Identify operations.
   - required ***operations*** are performed by calling methods.
4. Write an Algorithm.
   - arrange the objects and operations in an ***order*** that solves the problem

BINUS
UNIVERSITY
INTERNATIONAL

People
Innovation
Excellence

OOP in Python

BINUS UNIVERSITY INTERNATIONAL

People Innovation Excellence

First-class Everything

- Everything is a class in Python
  - This means that "everything" is treated the same way, everything is a class: functions and methods are values just like lists, integers or floats. Each of these are instances of their corresponding classes.

BINUS UNIVERSITY INTERNATIONAL

People Innovation Excellence

Important Terms

class
A user-defined compound type. A class can also be thought of as a template for the objects that are instances of it.

constructor
Every class has a “factory”, called by the same name as the class, for making new instances.

initializer method
A special method in Python (called __init__) that is invoked automatically to set a newly created object’s attributes to their initial (factory-default) state.

instance
An object whose type is of some class. Instance and object are used interchangeably.

BINUS UNIVERSITY INTERNATIONAL

People Innovation Excellence

Important Terms

instantiate
To create an instance of a class, and to run its initializer.

method
A function that is defined inside a class definition and is invoked on instances of that class.

object
A compound data type that is often used to model a thing or concept in the real world. It bundles together the data and the operations that are relevant for that kind of data. Instance and object are used interchangeably.

object-oriented programming
A powerful style of programming in which data and the operations that manipulate it are organized into objects.

object-oriented language
A language that provides features, such as user-defined classes and inheritance, that facilitate object-oriented programming.

BINUS UNIVERSITY INTERNATIONAL

People
Innovation
Excellence

Hands - on

1

We'll go through creating classes, instantiating objects, initializing attributes with the constructor method, and working with more than one object of the same class.

BINUS UNIVERSITY INTERNATIONAL

People
Innovation
Excellence

Hands - on

2

Demonstrate the use of both class and instance variables in object-oriented programming within Python

BINUS UNIVERSITY INTERNATIONAL

People
Innovation
Excellence

Hands - on

3

Understanding Inheritance in Python

BINUS UNIVERSITY INTERNATIONAL

People
Innovation
Excellence

Exercises....a TON of it!!!