Object-oriented programming

Lecture #5: Objects and Classes

Outline

- Objects
- Messages
- Classifications
- Classes
- Instances
- Meta-classes

Object-based Programming

- To view the world as a collection of autonomous, interacting objects
  - Examples: people, animals, plants, buildings, rooms, stairs, …

Real-World Object Properties

- Active, autonomous behaviors
  - Not directly controlled by the outside
- Communicative
  - Can send/receive “messages” to/from other objects
- Collaborative
  - long-term relationships between objects will arise

Real-World Object Properties

- Nested
  - Complex objects have other objects as components (which in turn may have object components ...)
- Uniquely named/identifiable
- Creation/Destruction
  - May be created and destroyed

Examples

- A person
  - Thinks (most of the time)
  - Communicates with others
  - Collaborates and works with others
  - Has a name (and a NRIC)
  - Is born/then dies

Examples

- A computer
  - Autonomous: it can do many things
  - Communicative (with people, with other computers)
  - A serial No.
  - Built/Destroyed

Virtual Objects

- Objects that exist in programs only:
  - Virtual objects consist of the above features
  - Virtual objects are much more precise in their names, borders, interactions, etc.
  - Virtual objects are the basic components for your object-oriented programs

Examples of Virtual Objects

- A bank account
  - Has a balance; responds to messages for deposits, withdrawals, and balance queries
- Set
  - Elements can be added, deleted, and queried

Examples of Virtual Objects

- E-Ticket
  - Records that customer has paid for service in advance of flight(s)
- Payment
  - A transaction in which money is exchanged

Virtual Object Example: a Set

- aSet understands the messages
  - aSet.add(anElement)
  - aSet.remove(anElement)
  - aSet.contains(anElement)
  - aSet.size()
- Messages have both an effect (causing internal changes or induced messages) and a return value
- Users only need to know external view of aSet to use it

Sets in non-OOP Languages

- A Set itself is a (passive) structure
- Operations on a Set are active but stateless functions, i.e., they do not remember anything from one call to the next
  - procedure add(s:Set, e:Element)

Sets in non-OOP Languages

- There is no encapsulation
  - The programmer could directly manipulate the data, possibly putting the data in a compromised state

What are Objects?

- Booch:
  - Something that “has state, behavior, and identity”
- Martin/Odell
  - “Anything, real, or abstract, about which we store data and those methods (operations) that manipulate the data”
- Peter Müller:
  - *An **object** is an instance of a class. It can be uniquely identified by its **name** and it defines a **state** which is represented by the values of its attributes at a particular time.*

Definition: Message

- A message is a request to an object to invoke one of its methods. A message therefore contains
  - the name of the method and
  - the arguments of the method
- Consequently, invocation of a method is just a reaction caused by receipt of a message. This is only possible if the method is actually known to the object

The Interface

- The operation “name” list open to other objects (other objects can send messages)
- If an object has a function but does not show it to the public, this is useless for the public (called private function)
- In C++, only public functions (belong to the interface) can be called by other objects (outsiders)

Properties of Objects

- Encapsulation (Data & Operations)
- Information Hiding
- Data Abstraction (with classes)
- Abstract Data Type (with classes)

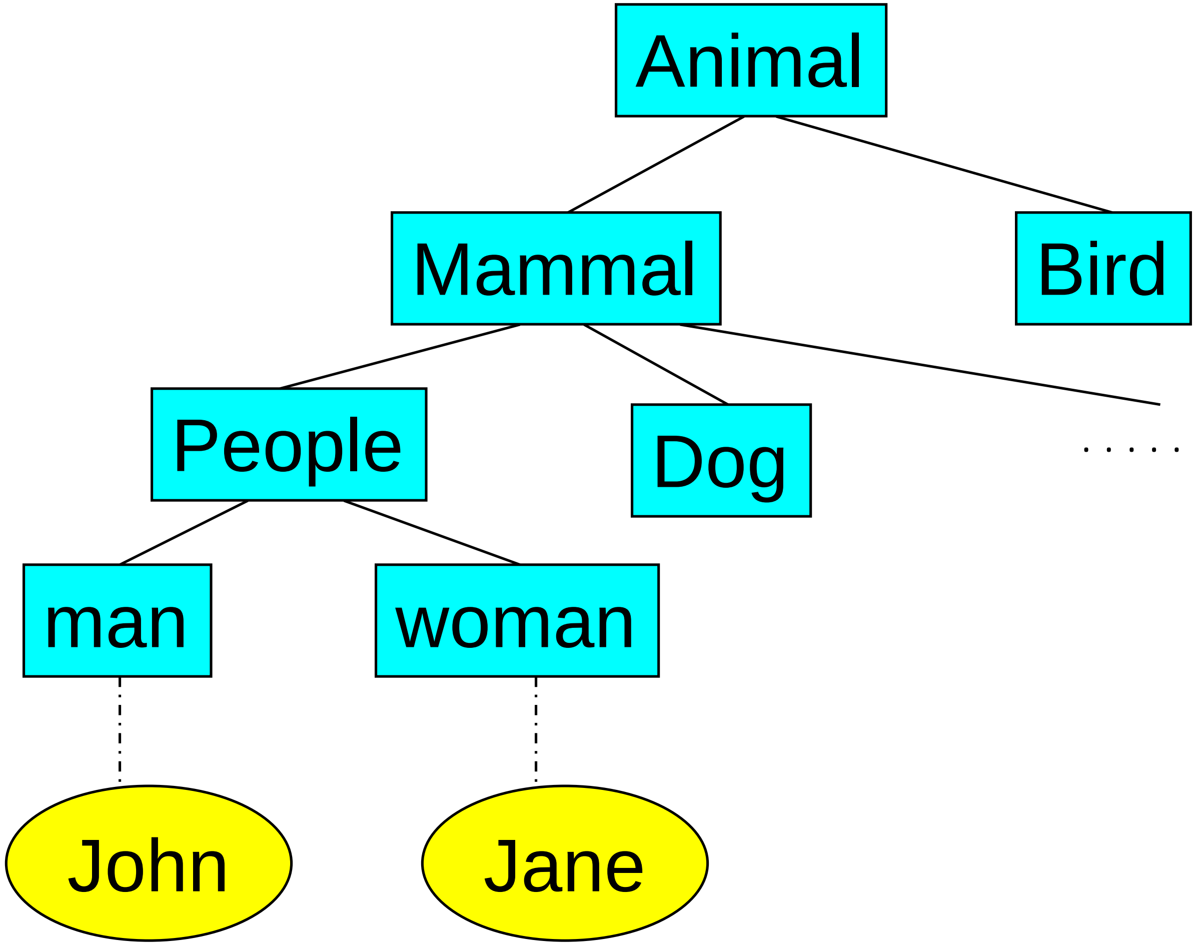
Advantages of Objects

- Same as Abstract Data Types
  - Information hiding
  - Data abstraction
  - Procedural abstraction
- Inheritance provides further data abstraction
  - Easier and less error-prone product development
  - Easier maintenance

Classification

- Classification is not unique to object-oriented systems. On the contrary, classification is applied in many other domains.

Classification



Class

- A class is either a classification or an abstraction of objects
- Class is not only a concept, but also a practical mechanism of programing
- OOP is actually programming with classes

Class

- “Class -- a definition of **an implementation** (methods and data structures) shared by a group objects.”
- “Class -- **a template** from which objects may be created. It contains a definition of the state descriptors and methods for the objects.”

Class

- Classes serve two distinct purposes
  - Factories that create new objects (code plus specifications)
  - Classification of objects (specification only: e.g. sizable class specifies any object with a size method)

Intentional Notion of Class

- New objects are instances of a class. Their states and behaviors are determined by the class definition
- This is so-called intentional notion of a class
- The intentional notion determines the structure of instances of that class

Extensional Notion of Class

- A class consists of object-warehouse and object-factory
- Object warehouse means that a class implicitly maintains a class extent
- A class extent consists of all instances of the class
- Object-factory means that there exists a constructor for each class, to generate new instances of that class

Understanding Classes

- A class provides a definition of the structure of instances of that class
- A class defines the names and attributes (state) and methods (behavior) of an object belonging to the class

Examples of Class

```
class Integer {
Ds:
    int I
Ops
    setValue(int n)
    Integer addValue(Integer j)
}
```

Examples of Class

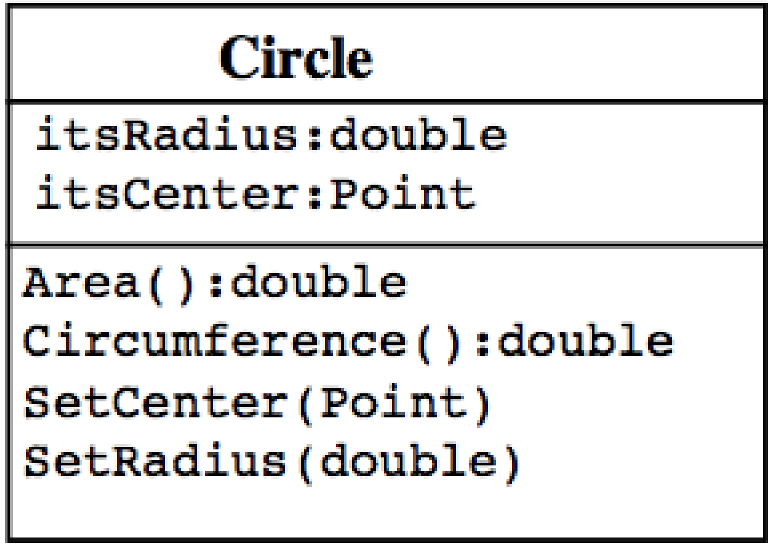
```
class Horse {
Ds:
  Age
  Weight
  Color
Ops:
  Drag
  Run
  Ride
}
```

Relationships among Classes

- Link (use-a)
  - A link relationship exists between two classes that need to communicate (an instance of one class sends a message to an instance of another class)
- Composition (has-a)
  - A composition relationship exists when a class contains data members that are also other class objects

UML: Representing Class

- UML stands for Unified Modeling Language



```
Circle
itsRadius:double
itsCenter:Point
Area():double
Circumference():double
SetCenter(Point)
SetRadius(double)
```

Instantiation

- The mechanism of creating new objects from a class definition is called instantiation
- Every class has such a mechanism

Instantiation

- Static instantiation and Dynamic instantiation
  - (static means) at compile time (by compiler);
  - (dynamic means) at run time
    - *Dynamic instantiation requires run-time support for allocation and de-allocation of memory*

Making Objects from Class Templates

- aSet = new Set()

/* Set is the class; this makes an object */

- anotherSet = new Set()

/* Same factory, but different contents */

- Each time an object is created, it is new and unique

Object

- Object ::= <Oid, Cid, Body>
  - Oid is the identification of the object;
  - Cid is the identification (or name) of the class of this object;
  - Body is the actual space for memory
- Note: the operations of the object are implemented in the class

Example of Class in C++

```
class Student {
private:
    unsigned numCoursesRequired;
    unsigned age;
public:
    Student(unsigned nCourses);
    void attendLecture();
    void selfStudy();
    void play();
};
```

The Relations between Class and Object

- An object is an instance of a class
- What if we treat class as an object???
- Then, what is the class of a class?
  - A meta-class

Meta-class

- A meta-class is the class of a class
- The attributes of a meta-class can be used to described a class (e.g. # of instances of a class)

Meta-class

- Explicit support of meta-classes means that objects, classes and meta-classes are treated uniformly
- Classes can be created at run-time by explicitly sending a message to a special meta-class
- Not all OOP languages support meta-class