Introduction To Object-Oriented Programming

Part I: You will learn how to define classes, create objects, call pre-defined methods.

James Tam

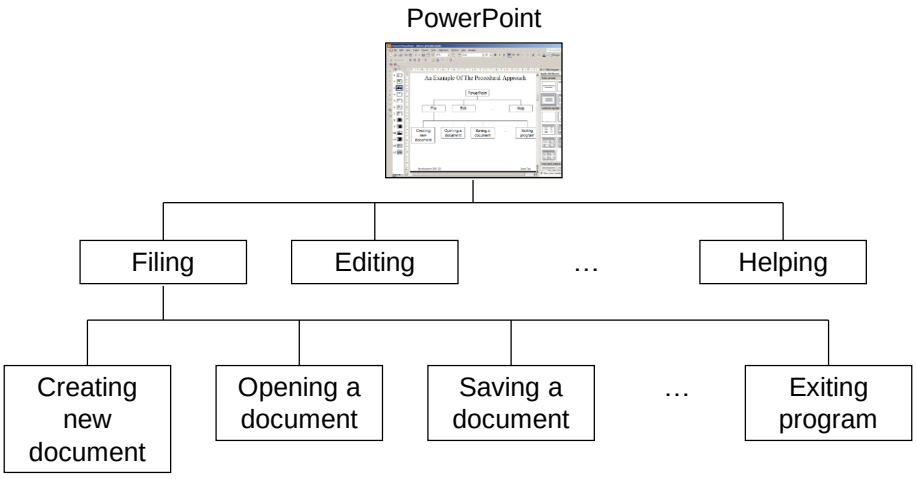
Reminder: What You Know

- There are different approaches to writing computer programs.
- They all involve decomposing your programs into parts.
- What is different between the approaches (how the decomposition occurs)/(criteria used for breaking things down")
- There approach to decomposition you have been introduced to thus far:
  - Procedural
  - Object-Oriented (~2 weeks for CPSC 231)

James Tam

An Example Of The Procedural Approach (Presentation Software)

- Break down the program by what it does (described with *actions/verbs*)



James Tam

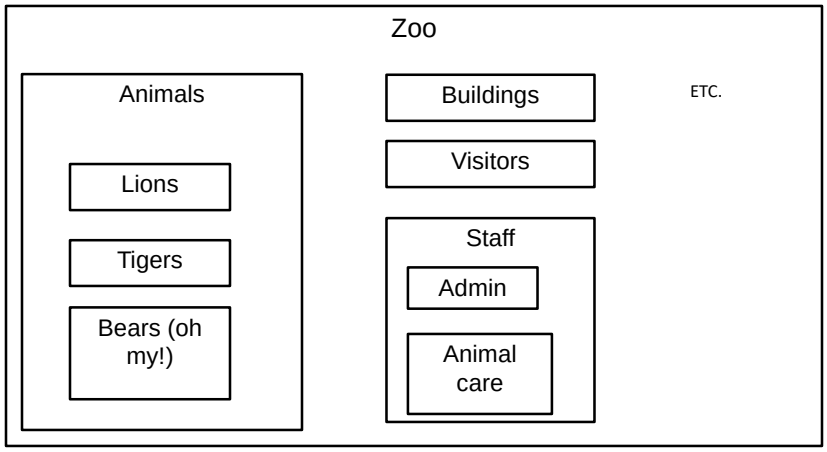
What You Will Learn

- How to break your program down into objects (**New term: "Object-Oriented programming"**)
- This and related topics comprise most of the remainder of the course

James Tam

An Example Of The Object-Oriented Approach (Simulation)

- Break down the program into entities (classes/objects - described with *nouns*)



James Tam

Classes/Objects

- Each class of object includes descriptive data.
  - Example (animals):
    - Species
    - Color
    - Length/height
    - Weight
    - Etc.
- Also each class of object has an associated set of actions
  - Example (animals):
    - Sleeping
    - Eating
    - Excreting
    - Etc.

James Tam

Example Exercise: Basic Real-World Alarm Clock

- What descriptive data is needed?
- What are the possible set of actions?

James Tam

Additional Resources

- A good description of the terms used in this section (and terms used in some of the later sections, last accessed 2021).
  http://docs.oracle.com/javase/tutorial/java/concepts/

- A good walk through of the process of designing an object-oriented program, finding the candidate objects e.g., how to use the "find a noun'"approach and some of the pitfalls of this approach , last accessed 2021).
  http://archive.eiffel.com/doc/manuals/technology/oosc/finding/page.html

James Tam

Types In Computer Programs

- Programming languages typically come with a built in set of types that are known to the translator

```
int num;
// 32 bit whole number (e.g. operations: +, -, *, /, %)

String s = "Hello";
// Unicode character information (e.g. operation:
concatenation)
```

- Unknown types of variables cannot be arbitrarily declared!

```
Person tam;
// What info should be tracked for a Person
// What actions is a Person capable of
// Compiler error! The identifier Person is unknown.
```

James Tam

A Class Must Be First Defined

- A class is a new type of variable.
- The class definition specifies:
  - What descriptive data is needed?
    - Programming terminology: **attributes = data** (**New definition**)
  - What are the possible set of actions?
    - Programming terminology: **methods = actions** (**new definition**)
    - A method is the Object-Oriented equivalent of a function

James Tam

Defining A Java Class

Format:

```
public class <name of class>
{
    attributes
    methods
}
```

Example (more explanations coming shortly):

```
public class Person
{
    private int age; // Attribute
    public Person() { // Method
        age = in.nextInt();
    }
    public void sayAge() {// Method
        System.out.println("My age is " + age);
    }
}
```

James Tam

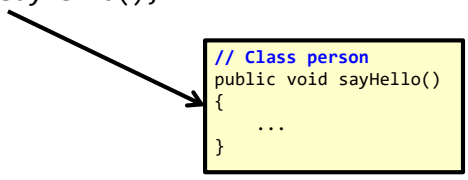
The First Object-Oriented Example

- Program design: each class definition (e.g., `public class <class name>`) must occur its own "dot-java" file).
- One example program consists of two files in the same directory:
  - (From now on your programs must be laid out in a similar fashion):
  - `Driver.java`
  - `Person.java`
  - **Full example is located in the folder**: `first_hello00`

James Tam

The Driver Class

```
public class Driver
{
    public static void main(String [] args)
    {
        Person aPerson = new Person();
        aPerson.sayHello();
    }
}
```




```
I don't wanna say hello.
```

James Tam

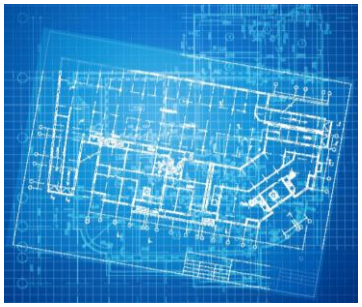
Class Person

```
public class Person
{
    public void sayHello()
    {
        System.out.println("I don't wanna say hello.");
    }
}
```

James Tam

New Concepts: Classes Vs. Objects

- Class definition:
  - Specifies the characteristics of an entity but is not an instance of that entity
  - It's much like a blue print that specifies the characteristics of a building (height, width, length etc.) a class definition only specifies attributes and methods but it doesn't create any examples (instances) of that class.

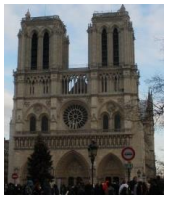


www.colorbox.com

James Tam

New Concepts: Classes Vs. Objects (2)

- Object:
  - A specific example or instance of a class.
  - Objects have all the attributes specified in the class definition



Images: James Tam

James Tam

`main()` Method

- Language requirement: There must be a `main()` method - or equivalent – to determine the starting execution point.
  - The main method has a very specific parameter list that must always be included (`String [] args`).
- Style requirement: the name of the class that contains `main()` is often referred to as the "`Driver`" class.
  - Makes it easy to identify the starting execution point in a big program.
  - An acceptable alternative name: Call it the "`Start`" class.
- Do not instantiate instances of the `Driver`[1]
- For now avoid:
  - Defining attributes for the `Driver`.[1]
  - Defining methods for the `Driver` (other than the `main()` method).[1]

1 Details may be provided later in this course

James Tam

Compiling Multiple Classes

- One way (safest) is to compile all code (dot-Java) files when any code changes.
- Example of what to type at the command line (with the first O-O example):
  - `javac Driver.java`
  - `javac Person.java`
  - (Alternatively use the 'wildcard'): `javac *.java`

James Tam

Why Must Classes Be Defined

- Some classes are already pre-defined (included) in a programming language with a list of attributes and methods e.g., `String`
- Why don't more classes come 'built' into the language?
- The needs of the program will dictate what attributes and methods are needed.

James Tam

Defining The Attributes Of A Class In Java

- Attributes can be variable or constant (preceded by the '`final`' keyword), for now stick to the former.
- **Format**:

  <access modifier>[1] *<type of the attribute> <name of the attribute>;*

- **Example**:

```
public class Person
{
    private int age;
}
```

1) Although other options may be possible, *attributes are almost always set to private* (more on this later). For now set all attributes to private.

James Tam

New Term: Object State

- Attributes: Data that describes each instance or example of a class.
- Different objects have the same attributes but the specific values contained in those attributes can vary.
  - Reminder: The class definition specifies the attributes and methods for *all objects.* Individual objects can have different values for attributes.
- Example: two 'monster' objects each have a health attribute but the current value of their health can differ.
- The current value of an object's attribute's determines it's state.

Age: 35
Weight: 192

Age: 50
Weight: 125

Age: 0.5
Weight: 7

James Tam

Defining The Methods Of A Class In Java

Format:

```
<access modifier>[1] <return type[2]> <method name> (<p1 type> <p1 name>, <p2 type>
 <p2 name>…)
 {
     <Body of the method>
 }
```

Example:

```
public class Person
{
    // Method definition
    public void sayAge() {
       System.out.println("My age is " + age);
    }
}
```

1) For now *set the access modifier on all your methods to 'public'* (more on this later).
2) Return types: includes all the built-in 'simple' types such as char, int, double…arrays and classes that have already been defined (as part of Java or third party extras). Void is for methods that don't have an explicit return statement or a specific return value.

James Tam

Defining Methods With Parameters: Different Types

Parameter type	Format	Example
Simple types	`<method>(<type> <name>)`	`method1(int x, char y) { ... }`
Objects	`<method>(<class> <name>)`	`method2(Person p) { ... }`
Arrays	`<method>(<type> []… <name>)`	`method3(Map [][] m) { ... }`

- When *calling* a method: Only the names of the parameters must be passed e.g., `System.out.println(num,age);`
- Multiple parameters are separated with a comma.

James Tam

Defining Methods, Specifying Return Values: Different Types

Return type	Example
Simple types	`int method1() { return(0); }`

Objects	`Person method2() {`
	`  Person p = new Person();`
	`  return(p);`
	`}`

Arrays	`Person [] method3() {`
	`  Person [] p = new`
	`  Person[3];`
	`  return(p);`
	`}`

Nothing	`Person void method4() {`
	`  ...`
	`  if (age < 0) {`
	`  return;`
	`  }`
	`  else { //Process the age variable.`
	`  }`
	`}`

James Tam

What Are Methods

- Possible behaviors or actions that apply to *each* instance (example) of a class.

Walk()
Talk()

Walk()
Talk()

Fly()

Swim()

James Tam

Instantiation

- **New definition**: Instantiation, creating a new instance or example of a class.
- Instances of a class are referred to as objects.
- **Format**:

```
<class name> <instance name> = new <class name>(<parameters>);
```

- **Examples**:

```
Person jim = new Person();
Scanner in = new Scanner(System.in);
```

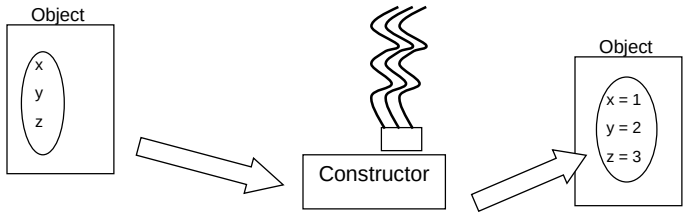
Creates new object

Reference variable names: 'jim', 'in'

James Tam

Constructor

- **New term**: A special method used to initialize the attributes of an object as the objects are instantiated (created).



- The constructor is automatically invoked whenever an instance of the class is created e.g., `Person aPerson = new Person();`

Call to constructor (creates something 'new')

```
class Person {
    // Constructor
    public Person() {
        …
    }
}
```

- Constructors can take parameters but **never** have a return type.

James Tam

New Term: Default Constructor

- Takes no parameters
- If no constructors are defined for a class then a default constructor comes 'built-into' the Java language for every class that you define.
- e.g.,

```
public class Driver {
    main() {
        Person aPerson = new Person();
    }
}

public class Person {
    private int age;
}
```

James Tam

Calling Methods (Outside The Class)

- You've already done this before with pre-created classes!
- First create an object (illustrated in previous screens)
- Then call the method for a particular variable.
- **Format**:

  <instance name>.<method name>(<p1 name>, <p2 name>…);

- **Examples:**

```
Person jim = new Person();
jim.sayName();

// Previously covered example, calling Scanner class method
Scanner in = new Scanner(System.in);
System.out.print("Enter your age: ");
age = in.nextInt();
```

Scanner variable

Calling method

James Tam

Calling Methods: Inside The Class

- You have seen this implicitly in the examples but here are the explicit syntax requirements you need to know well.
- Calling a method inside the body of the class (where the method has been defined)
  - You can just directly refer to the method (or attribute)

```
public class Person {
  private int age;

  public void birthday() {
     becomeOlder();  // access a method
  }

 public void becomeOlder() {
     age++;    // access an attribute
 }
```

James Tam

Calling Methods: Outside The Class

- Calling a method outside the body of the class (i.e., in another class definition).
- The method must be prefaced by a variable (actually a reference to an object – more on this later).

```
public class Driver {
    public static void main(String [] args) {
        Person bart = new Person();
        Person lisa = new Person();
        // Incorrect! Who ages?
        becomeOlder();

        // Correct. Happy birthday Bart!
        bart.becomeOlder();
    }
}
```

James Tam

Second Object-Oriented Example

- Learning concepts:
  - Attributes
  - Constructors
  - Accessing class attributes in a class method
- **Full example is located in the folder**: `second_attributeConstructor`

James Tam

Class Driver

```
public class Driver
{
    public static void main(String [] args)
    {
        Person jim = new Person();
        jim.sayAge();
    }
}
```

```
public Person() {
    Scanner in = new
      Scanner(System.in);
    System.out.print("Enter age: ");
    age = in.nextInt();
 }
```

```
public void sayAge() {
    System.out.println
      ("My age is " + age);
 }
```

```
[csc firstOOExample 232 ]> java Driver
Enter age: 123
My age is 123
```

```
[csc firstOOExample 233 ]> java Driver
Enter age: 321
My age is 321
```

James Tam

Class Person

```
public class Person
{
    private int age;
    public Person()
    {
        Scanner in = new Scanner(System.in);
        System.out.print("Enter age: ");
        age = in.nextInt();
    }

    public void sayAge()
    {
        System.out.println("My age is " + age);
    }
}
```

James Tam

Creating An Object

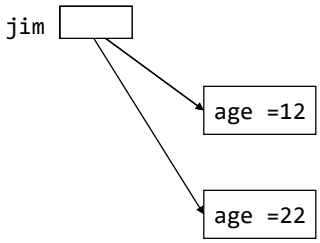
- Two stages (can be combined but don't forget either step)
  - Create a variable that refers to an object e.g., `Person jim;`
  - Create a *new* object e.g., `jim = new Person();`
    - The keyword 'new' calls the constructor to create a new object in memory
  - Observe the following

```
Person jim;

jim = new Person(12);

jim = new Person(22);
```

Jim is a reference to a Person object



James Tam

New Terms And Definitions

- Object-Oriented programming
- Class
- Object
- Class attributes
- Class methods
- Object state
- Instantiation
- Constructor (and the Default constructor)

James Tam

After This Section You Should Now Know

- How to define classes, instantiate objects and access different part of an object (attributes, methods).
- What is a constructor and how is it defined and used.

James Tam

Copyright Notification

- "Unless otherwise indicated, all images in this presentation are used with permission from Microsoft."

James Tam