Chapter 1.

Introduction

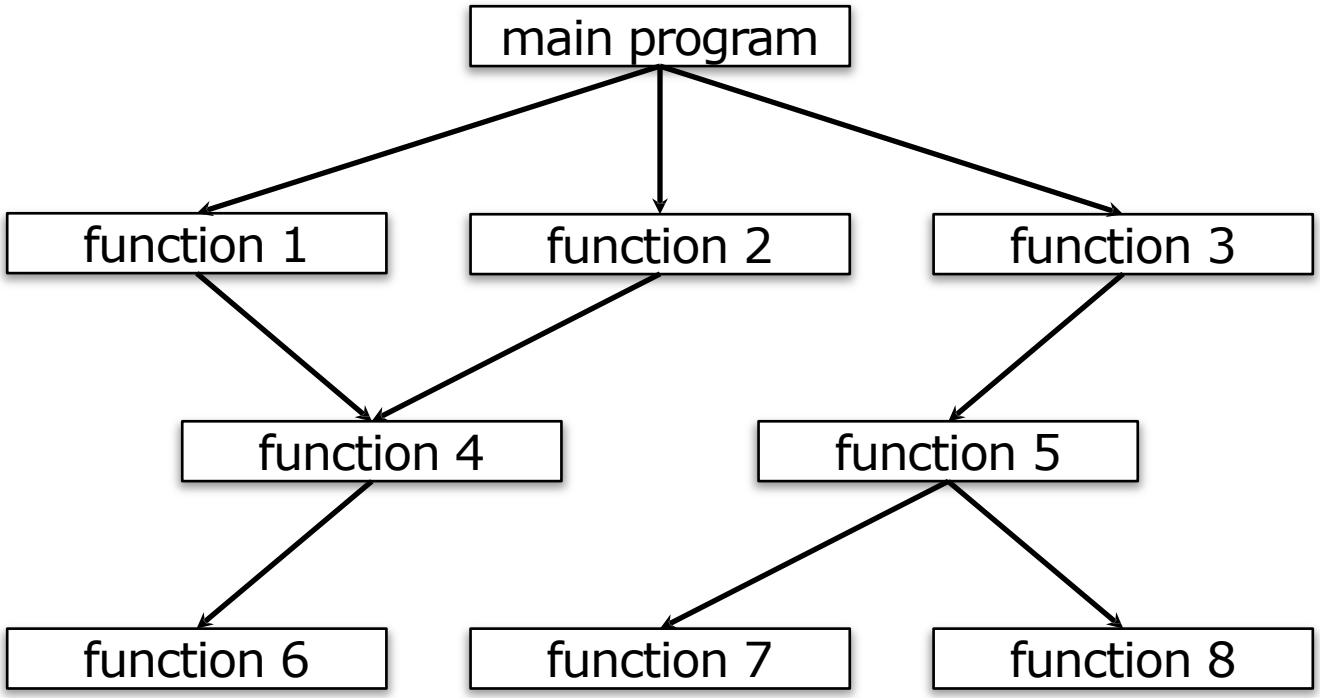
2020-2021
COMP2396 Object-Oriented Programming and Java
Dr. T.W. Chim (E-mail: twchim@cs.hku.hk)
Department of Computer Science, The University of Hong Kong

Procedural Programming

- Emphasis is on tasks to be done
- Break down a problem into a number of tasks and sub-tasks (top-down design approach)
- Implement tasks and sub-tasks as functions
- A function is simply a group of instructions that are executed when the function is being called
- A program is composed of a collection of functions that operate on some (shared) data

Procedural Programming

- Typical structure of a program in procedural programming



Procedural Programming

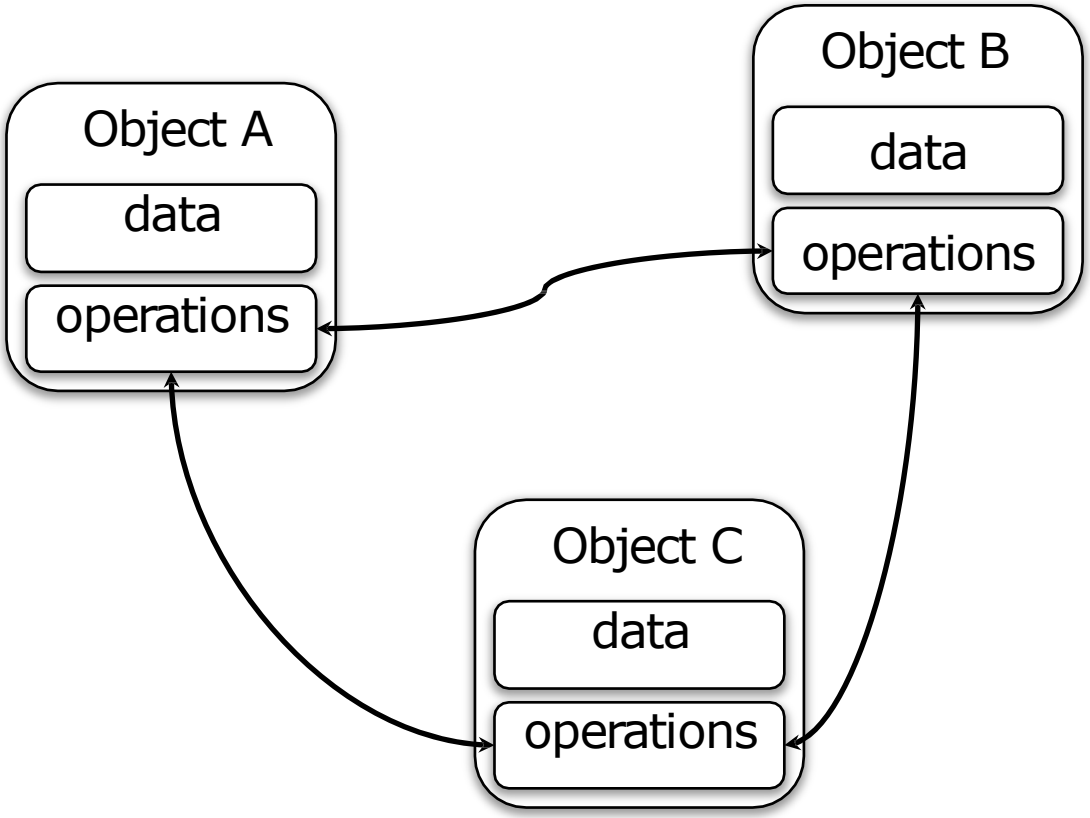
- Data may be accessed by multiple functions and become vulnerable (hard to trace when something goes wrong!)
- Difficult to maintain and extend (e.g., changing a data structure involves modifying all functions that work with that data structure)
- Code reusability is low

Object-Oriented Programming

- Emphasis is on data
- Decompose a problem into a number of entities called objects (bottom-up design approach)
- Implement objects using classes
- A class is like a blueprint for an object, it defines the state (data) and behavior (operations) supported by an object
- A program is composed of a collection of objects which interact with each other

Object-Oriented Programming

- Typical structure of a program in object-oriented programming



Object-Oriented Programming

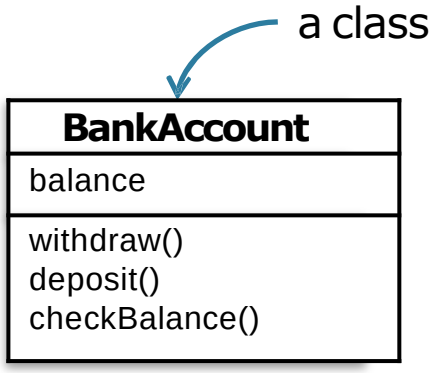
- Data of an object can be accessed only by operations associated with that object (protected from external access, easy to trace)
- Operations of an object can, however, access the operations of other objects (object interaction)
- Easy to maintain and extend (e.g., changing a data structure inside a class involves only modifying operations of that class which work with that data structure)
- Code reusability is high (through reusing existing classes or deriving new classes from existing classes)

Objects in OOP

- An object has both state (data) and behavior (operations)
- Data of an object are stored in its instance variables
- Operations that an object can perform are called its methods
- Objects can be used to model real world objects (e.g., car, dog), as well as concepts (e.g., date, bank account)
- An object instantiated (created) from a class is called an instance of that class

Objects in OOP

- Instances of the same class
  - Have the same set of instance variables that represent their states (e.g., balance of a bank account)
  - The actual values stored in their instance variables may be different
  - Have the same set of methods (e.g., withdrawing money from a bank account, depositing money into a bank account, checking the balance of a bank account)
  - Their actual behaviors, however, depend on their own states (e.g., withdrawing money from a bank account with a non-positive balance may not be successful)

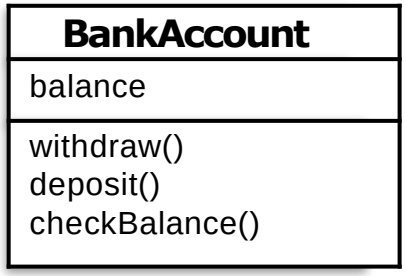


Fundamental OOP Concepts

- Abstraction
  - The act of representing essential features without including background details
- Encapsulation
  - The act of wrapping up of data and operations into a single unit
- Inheritance
  - The process by which objects of one class acquire the properties of objects of another class
- Polymorphism
  - The ability to take more than one form

Abstraction

- The act of representing essential features without including background details
- Focus on what an object is or does rather than how it is represented or how it works
- Provide an abstract description of an object by taking away unimportant details (problem dependent!)
- Model only necessary and common properties of objects (e.g., all types of bank accounts maintain a balance, and support methods like withdrawing money, depositing money and checking balance)
- Essential in the design of classes and objects

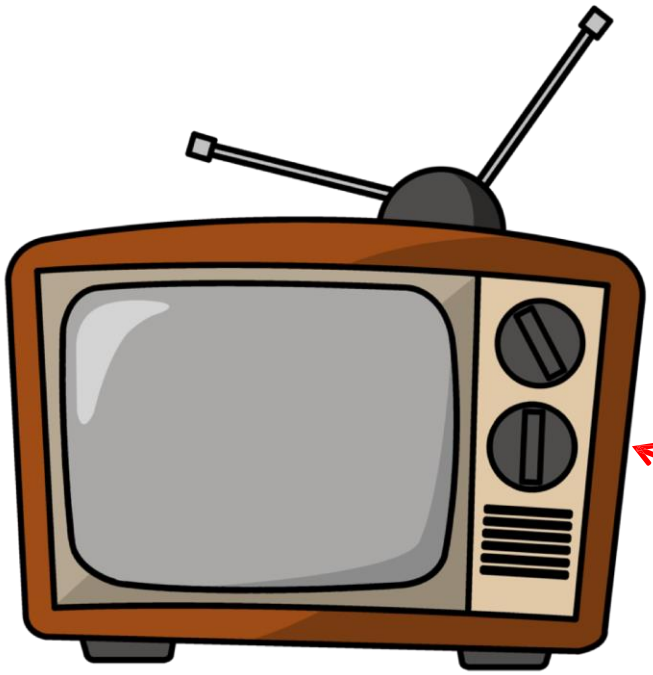


Encapsulation

- The act of wrapping up of data and operations into a single unit (i.e., an object)
- Hide the actual implementations (of both data and operations) from the outside world
- Data cannot be accessed directly from outside an object (information hiding)
- Interactions with an object only through its exposed interface (public methods)
- Allow changing internal implementations of an object without hurting the overall functioning of the system
- Essential in the implementation of classes and objects

A Real World Example

- Abstraction and encapsulation

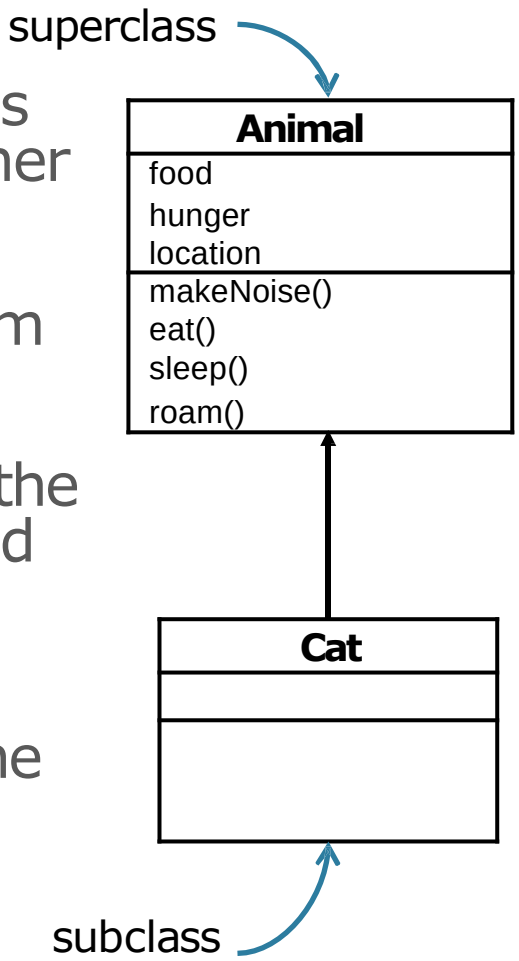


Television
listOfChannels currentChannel currentVolume onOffState
channelUp() channelDown() volumeUp() volumeDown() switchOn() switchOff()

The actual implementations (e.g., circuit logic and electronic components) are hidden inside the case

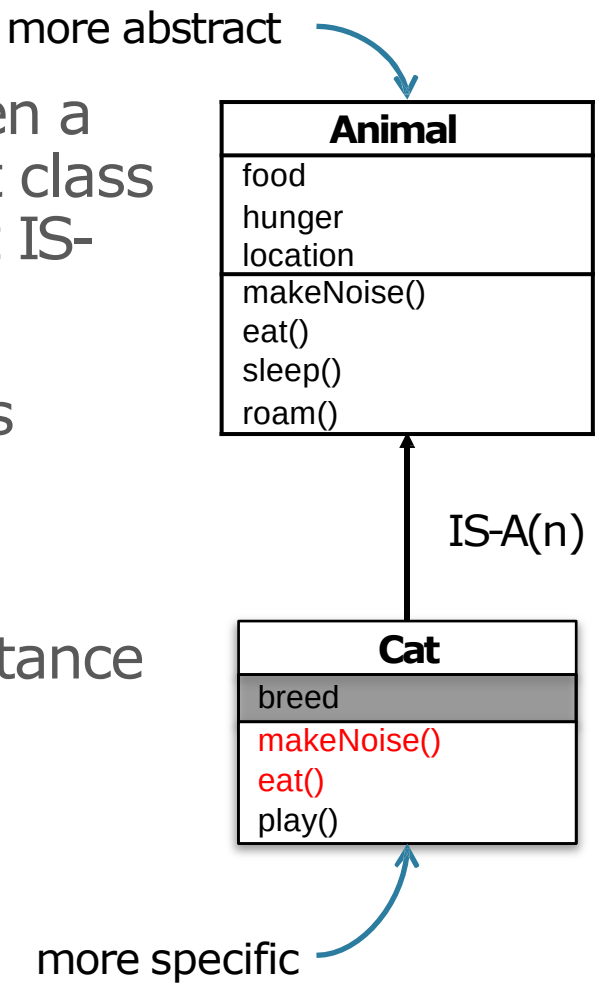
Inheritance

- The process by which objects of one class acquire the properties of objects of another class
- OOP allows a new class to be derived from an existing class
- The new class is known as a subclass of the existing class from which it is derived, and the existing class is referred to as the superclass of the derived class
- A subclass will automatically inherit all the properties (both instance variables and methods) of its superclass



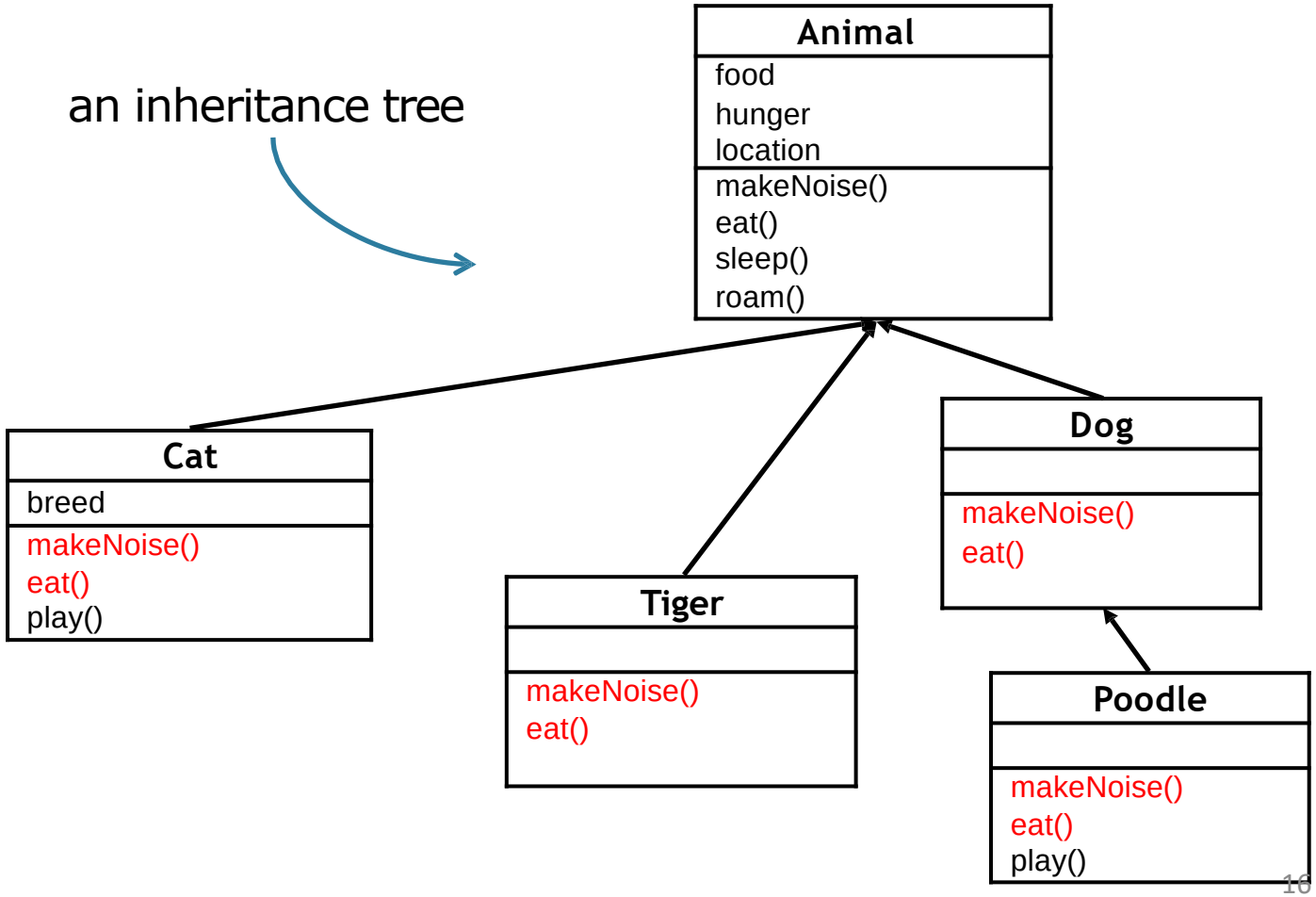
Inheritance

- There exists a IS-A relationship between a subclass and its superclass (e.g., if Cat class is a subclass of Animal class, then Cat IS-A(n) Animal)
- A subclass can override methods of its superclass by providing its own implementations of the methods
- A subclass can also introduce new instance variables as well as new methods
- A subclass can be considered as a specialization of its superclass



Inheritance

- Support the concept of hierarchical classification



Polymorphism

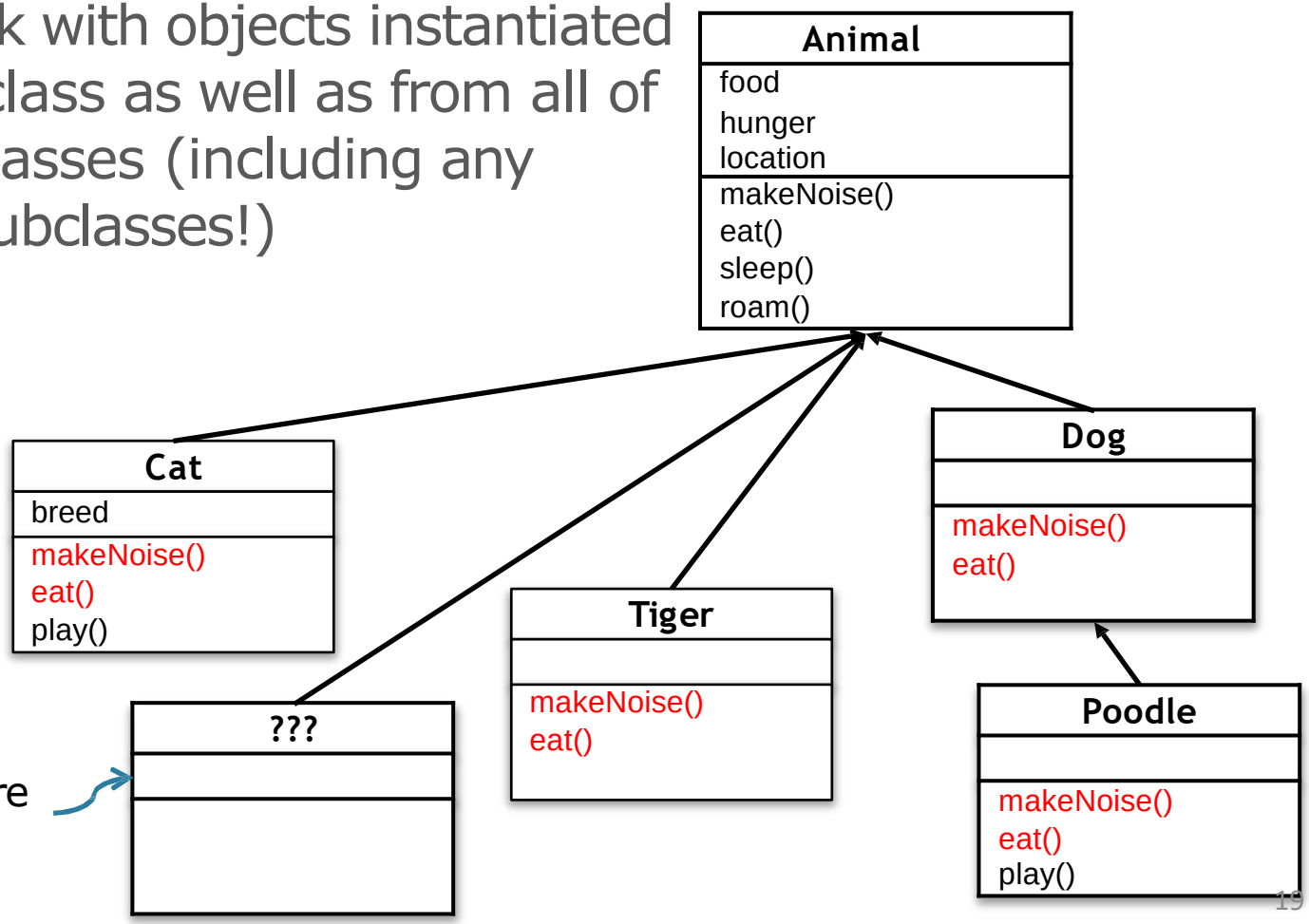
- The ability to take more than one form
- OOP allows a subclass object be used in place of its superclass object (IS-A relationship!)
- Examples
  - A method taking an Animal object as its argument can also take a Cat object as its argument given Cat class is a subclass of Animal class (Cat IS-A(n) Animal!)
  - An array declared to hold Animal objects can also hold a Cat object or a Dog object given both Cat class and Dog class are subclasses of Animal class

Polymorphism

- Consider objects instantiated from different subclasses (e.g., Cat class & Dog class) of a common superclass (e.g., Animal class)
- They all have identically named methods (overridden or not) inherited from their common superclass (e.g., makeNoise(), eat())
- Making a call to the same method (e.g., makeNoise()) of these objects will result in different behaviors depending on the actual subclass object being called

Polymorphism

- Support designing methods that can work with objects instantiated from a class as well as from all of its subclasses (including any future subclasses!)



Java

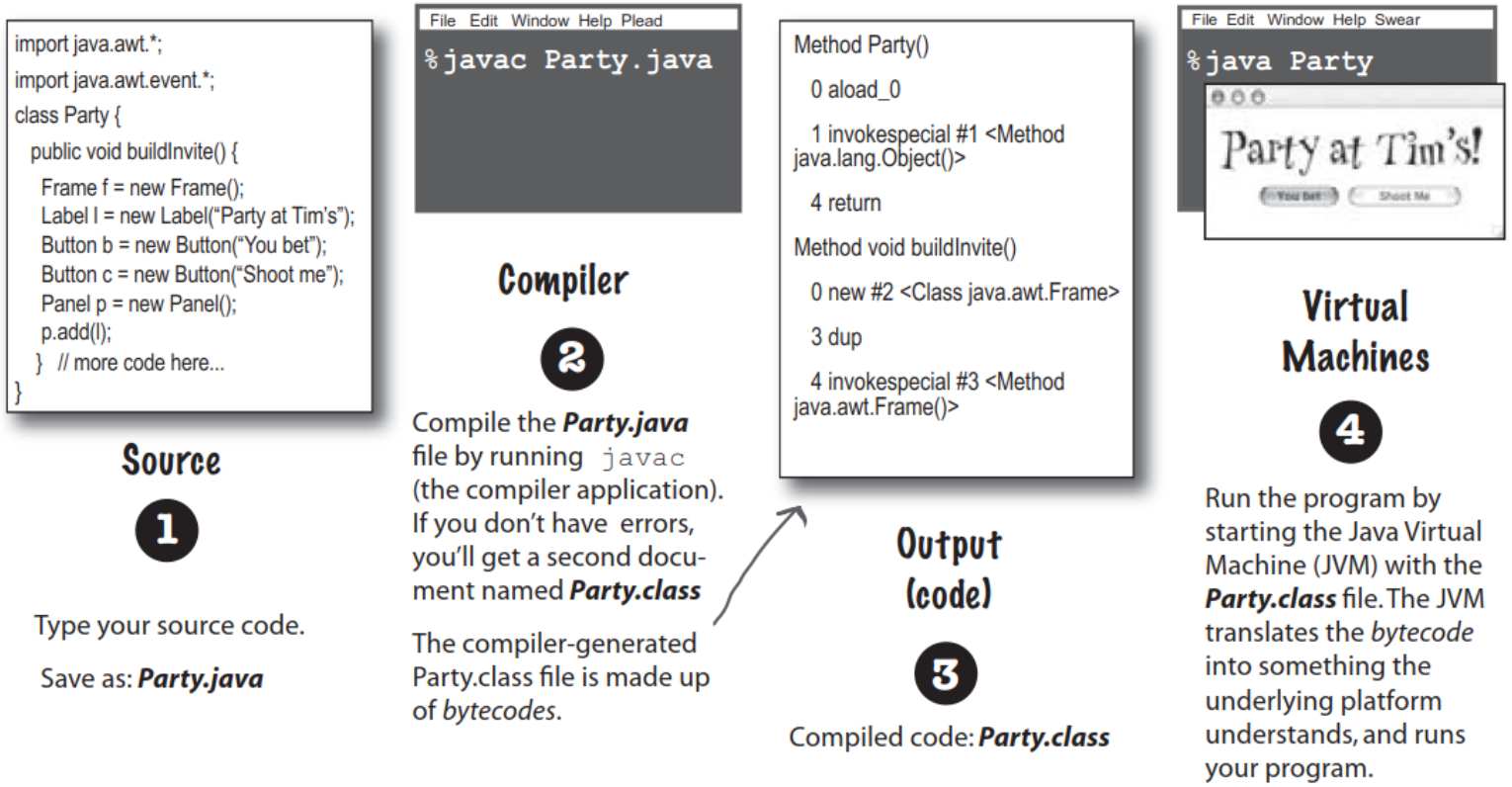
- A programming language that is
  - Simple – Java is designed to be easy to learn. Syntax is similar to C/C++, with some features (e.g., structure, pointer) removed
  - Object-oriented – In Java, everything is an object
  - Platform independent – A Java program is compiled into platform independent byte code that can be executed by virtual machines on different platforms
  - Architectural-neutral – Java compiler generates an architecture-neutral object file format which makes the compiled code to be executable on many processors

Java

- A programming language that is (continue…)
  - Portable – Being architectural-neutral and having no implementation dependent aspects of the specification make Java portable
  - Robust – With compile time error checking and runtime error checking
  - Multi-threaded – it is possible to write programs that can do many tasks simultaneously
  - High performance, secure, distributed, dynamic…

Write Once, Run Anywhere

- Steps to produce and run a Java program



Code Structure in Java

- Put a class in a source file

A source code (*.java) holds one class definition. The class represents a piece of your program. The body of the class must go within a pair of curly braces.

```
public class Example {
   void method1() {
      statement1;
      statement2;
   }
   void method2() {
         statement3;
         statement4;
         statement5;
   }
}
```

Example.java

Code Structure in Java

- Put methods in a class

A class has one or more methods. All methods must be declared inside a class

```
public class Example {
   void method1() {
      statement1;
      statement2;
   }
   void method2() {
         statement3;
         statement4;
         statement5;
   }
}
```

Example.java

Code Structure in Java

- Put statements in a method

A method holds statements (instructions) for how that particular method should be performed

```
public class Example {
   void method1() {
      statement1;
      statement2;
   }
   void method2() {
         statement3;
         statement4;
         statement5;
   }
}
```

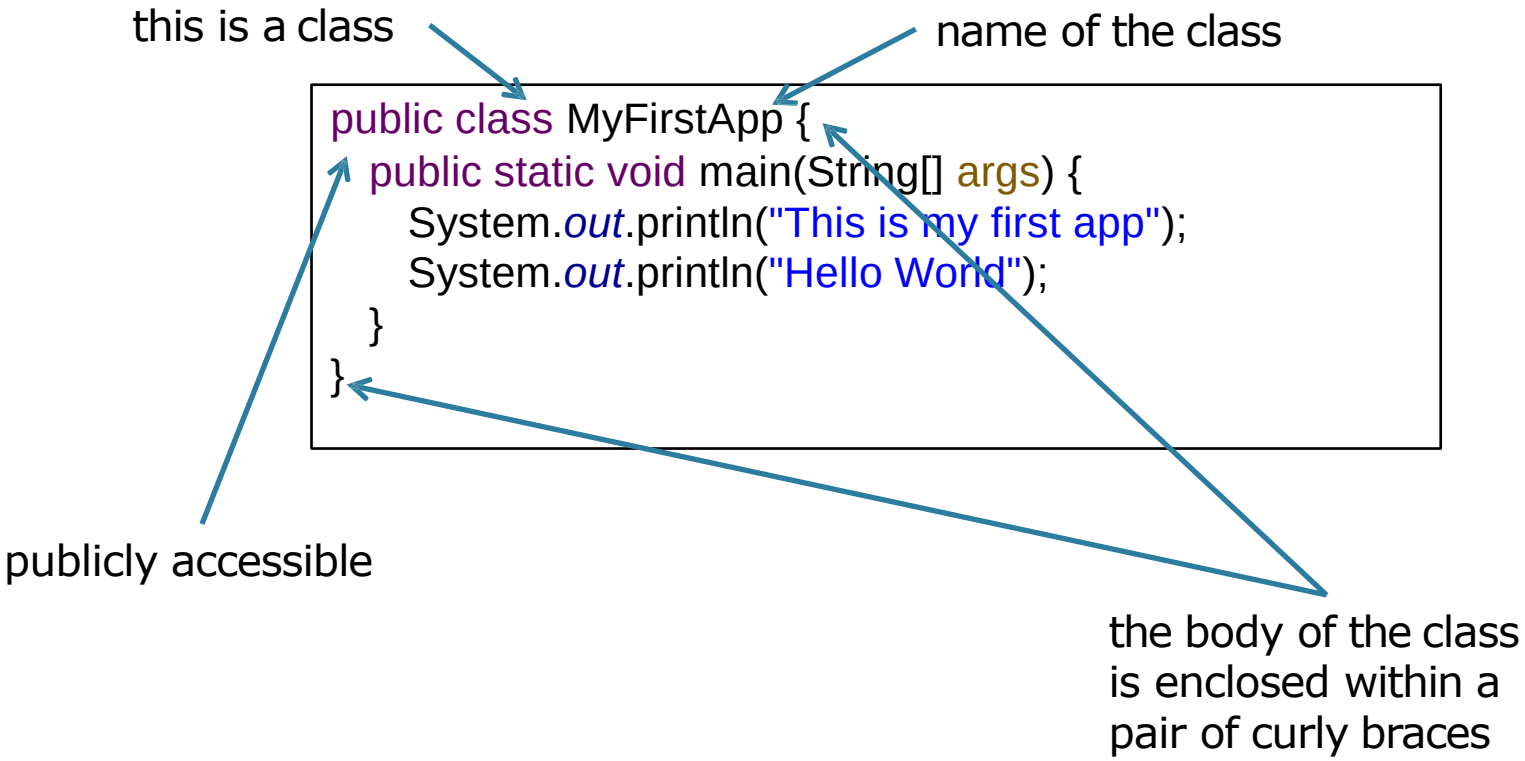
Example.java

My First Java Program

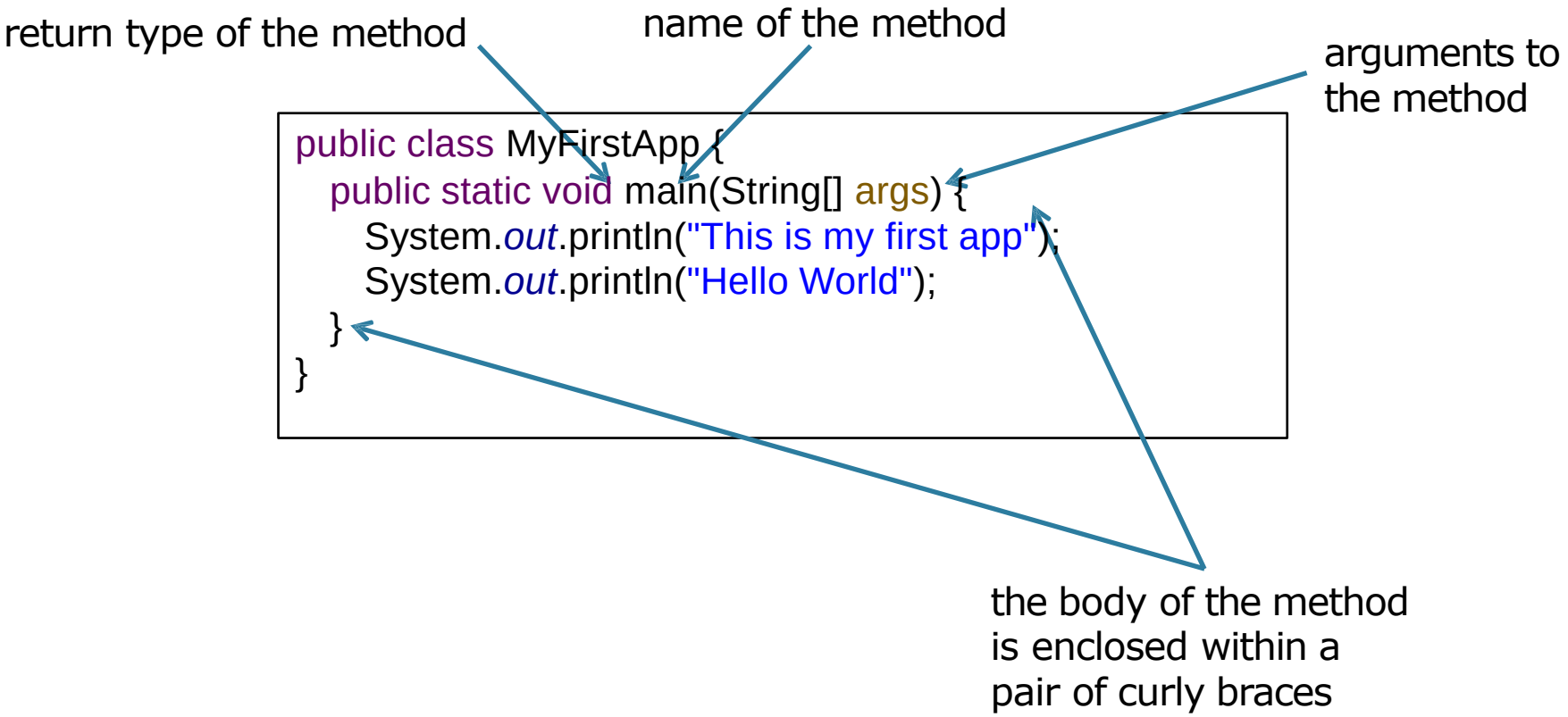
```
public class MyFirstApp {
   public static void main(String[] args) {
      System.out.println("This is my first app");
      System.out.println("Hello World");
   }
}
```

- Every Java application has to have at least one class
- The main Java class (that is the class you call using "java") has to have one main() method

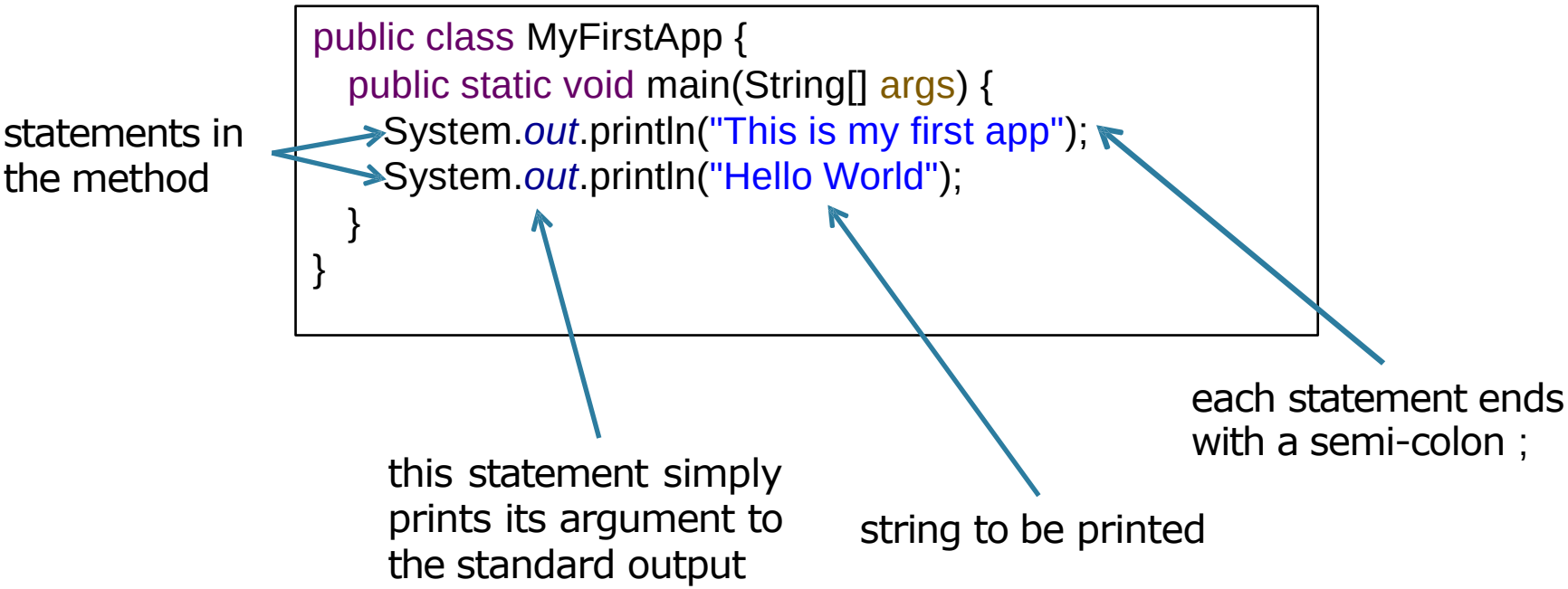
My First Java Program



My First Java Program



My First Java Program



My First Java Program

```
public class MyFirstApp {
   public static void main(String[] args) {
      System.out.println("This is my first app");
      System.out.println("Hello World");
   }
}
```

1. Type the source code and save it as MyFirstApp.java
2. Compile it into MyFirstApp.class
3. Run the program under Java Virtual Machine (JVM)

My First Java Program

```
public class MyFirstApp {
   public static void main(String[] args) {
      System.out.println("This is my first app");
      System.out.println("Hello World");
   }
}
```

- When the Java Virtual Machine (JVM) starts running, it will first load the MyFirstApp class
- Next the JVM will look for the main() method of the MyFirstApp class and start executing it

Statements in a Method

- Declarations, assignments, method calls, etc.

```
int x = 3;
String name = "OOP";
x = x * 17;
System.out.print("x is " + x);
double d = Math.random();
```

- Branching

```
if (x == 10) {
   System.out.print("x must be 10");
} else {
   System.out.print("x isn't 10");
}
```

Statements in a Method

- Loops

```
int x = 0;
while (x < 10) {
   System.out.println("x is now " + x);
   x = x + 1;
}
```

```
for (int x = 0; x < 10; x = x + 1) {
   System.out.println("x is now " + x);
}
```

- Comment lines

```
// this is a comment line

/* this is a comment
     spanning  multiple lines */
```

Boolean Test in Java

- Note that a boolean and an integer are not compatible types in Java
- Hence the following boolean test is illegal in Java

```
int x = 1;
while (x) { … }
```

- The only variable that can be directly tested (without using a comparison operator) is a boolean

```
boolean isHot = true;
while (isHot) { … }
```

Example: Phrase-O-Matic

- What does the following application do?

```
public class PhraseOMatic {
   public static void main(String[] args) {

      String[] girls = {"Amanda", "Jessica", "Chrissie"};
      String[] verbs = {"loves", "hates"};
      String[] boys = {"Andy", "Aaron", "Eason", "Jacky"};

      int i = (int) (Math.random() * girls.length);
      int j = (int) (Math.random() * verbs.length);
      int k = (int) (Math.random() * boys.length);

      System.out.print(girls[i] + " " + verbs[j] + " " + boys[k]);

   }
}
```

Example: Phrase-O-Matic

- What does the following application do?

```
public class PhraseOMatic2 {
   public static void main(String[] args) {

      String[] girls = {"Amanda", "Jessica", "Chrissie"};
      String[] verbs = {"loves", "hates"};
      String[] boys = {"Andy", "Aaron", "Eason", "Jacky"};

      for (int i = 0; i < girls.length; i++) {
         int j = (int) (Math.random() * verbs.length);
         int k = (int) (Math.random() * boys.length);
         System.out.println(girls[i] + " " + verbs[j] + " " + boys[k]);
      }

   }
}
```

Example: Phrase-O-Matic

- What does the following application do?

```
public class PhraseOMatic3 {
   public static void main(String[] args) {

      String[] girls = {"Amanda", "Jessica", "Chrissie"};
      String[] verbs = {"loves", "hates"};
      String[] boys = {"Andy", "Aaron", "Eason", "Jacky"};

      for (String girl : girls) { // for each string in girls[]
         int j = (int) (Math.random() * verbs.length);
         int k = (int) (Math.random() * boys.length);
         System.out.println(girl + " " + verbs[j] + " " + boys[k]);
      }

   }
}
```

We are with you!

If you encounter any problems in understanding the materials in the lectures, **please feel free to contact me or my TAs**.
We are always with you!

We wish you enjoy learning Java in this class.

Chapter 1.

End

2020-2021
COMP2396 Object-Oriented Programming and Java
Dr. T.W. Chim (E-mail: twchim@cs.hku.hk)
Department of Computer Science, The University of Hong Kong