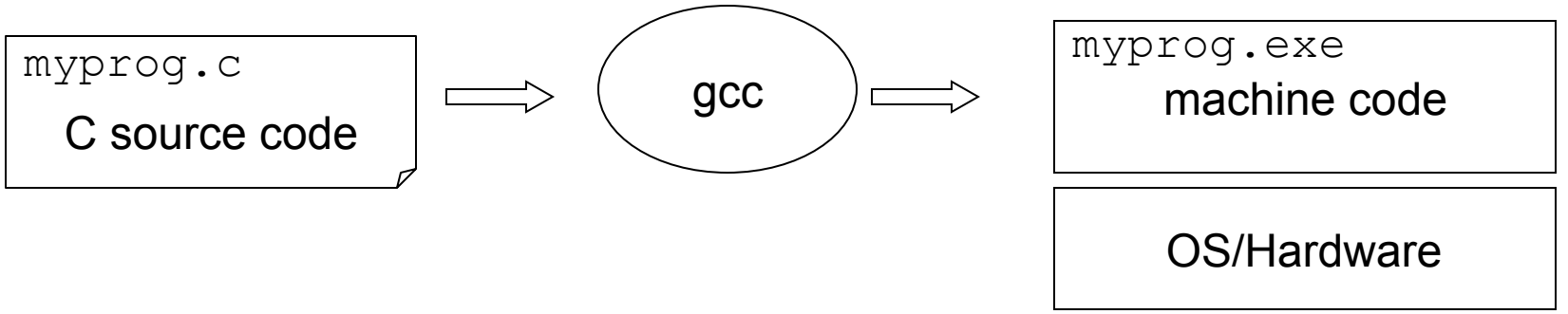
Big Data
- Java basics

Dr. Qing “Matt” Zhang
ITU

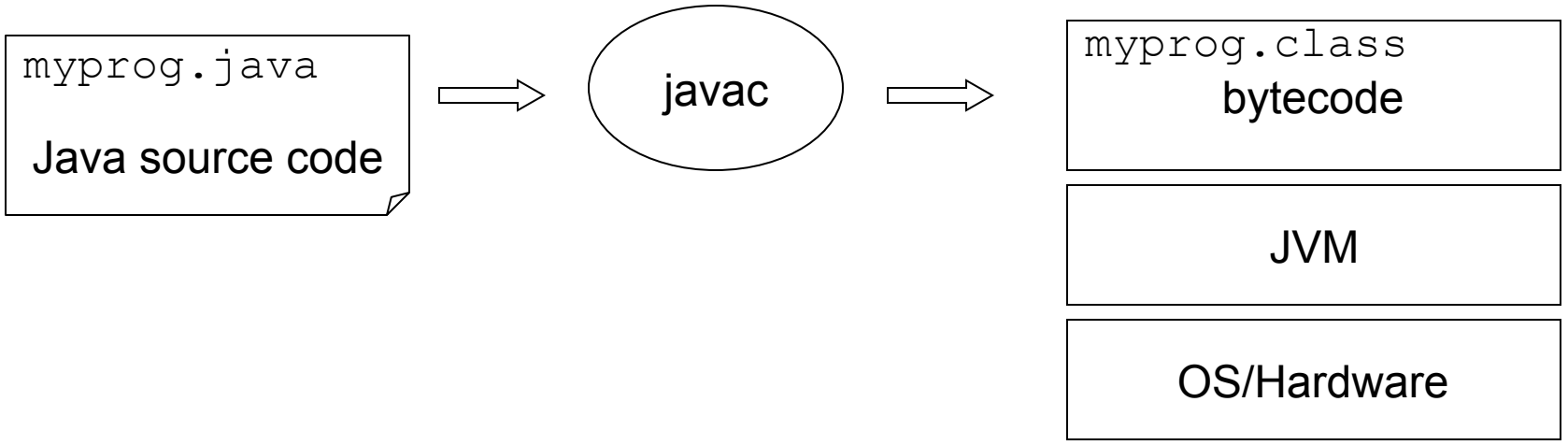
Java Intro

- Platform independent
  - Code is compiled into bytecode
  - Interpreted by JVM
  - Portable across platforms
- Security
- Garbage collection

Platform Dependent



Platform Independent



Hello World

- Hello.java

```
/*
This program prints out "hello world!" and terminates.
*/
public class Hello {
        public static void main (String args[]) {

                System.out.println("Hello world!");

        } // end of main
} // end of class
```

Compiling and Running

In order to run a Java program:

- First you compile it
  - that is, you run a program called **compiler** that checks whether the program follows the Java syntax
  - if it finds errors, it lists them
  - If there are no errors, it translates the program into Java bytecode
  - Example: `javac Hello.java`
  - If successful, this creates a file Hello.class which contains the translation (Java bytecode) of Hello.java
- Then you execute it
  - That is, you call the Java Virtual Machine to interpret and execute the Java bytecode of your program
  - Example:

    `java Hello`

Syntax

- Comments
  - what follows after // on the same line is considered comment
  - Or, what is in between /* this is a comment */
- Indentation
  - is for the convenience of the reader; compiler ignores all spaces and new lines ; the delimiter for the compiler is the semicolon
- All instructions end by semicolon
- Lower vs. upper case matters!!
  - Void is different than void
  - Main is different that main

Primitive Data Types

Data Type	Characteristics	Range
`byte`	8 bit signed integer	-128 to 127
`short`	16 bit signed integer	-32768 to 32767
`int`	32 bit signed integer	-2,147,483,648 to 2,147,483,647
`long`	64 bit signed integer	-9,223,372,036,854,775,808 to-9,223,372,036,854,775,807
`float`	32 bit floating point number	± 1.4E-45 to ± 3.4028235E+38
`double`	64 bit floating point number	± 4.9E-324 to ± 1.79769313486231 57E+308
`boolean`	`true` or `false`	NA, note Java booleans cannot be converted to or from other types
`char`	16 bit, Unicode	Unicode character, `\u0000` to `\uFFFF` Can mix with integer types

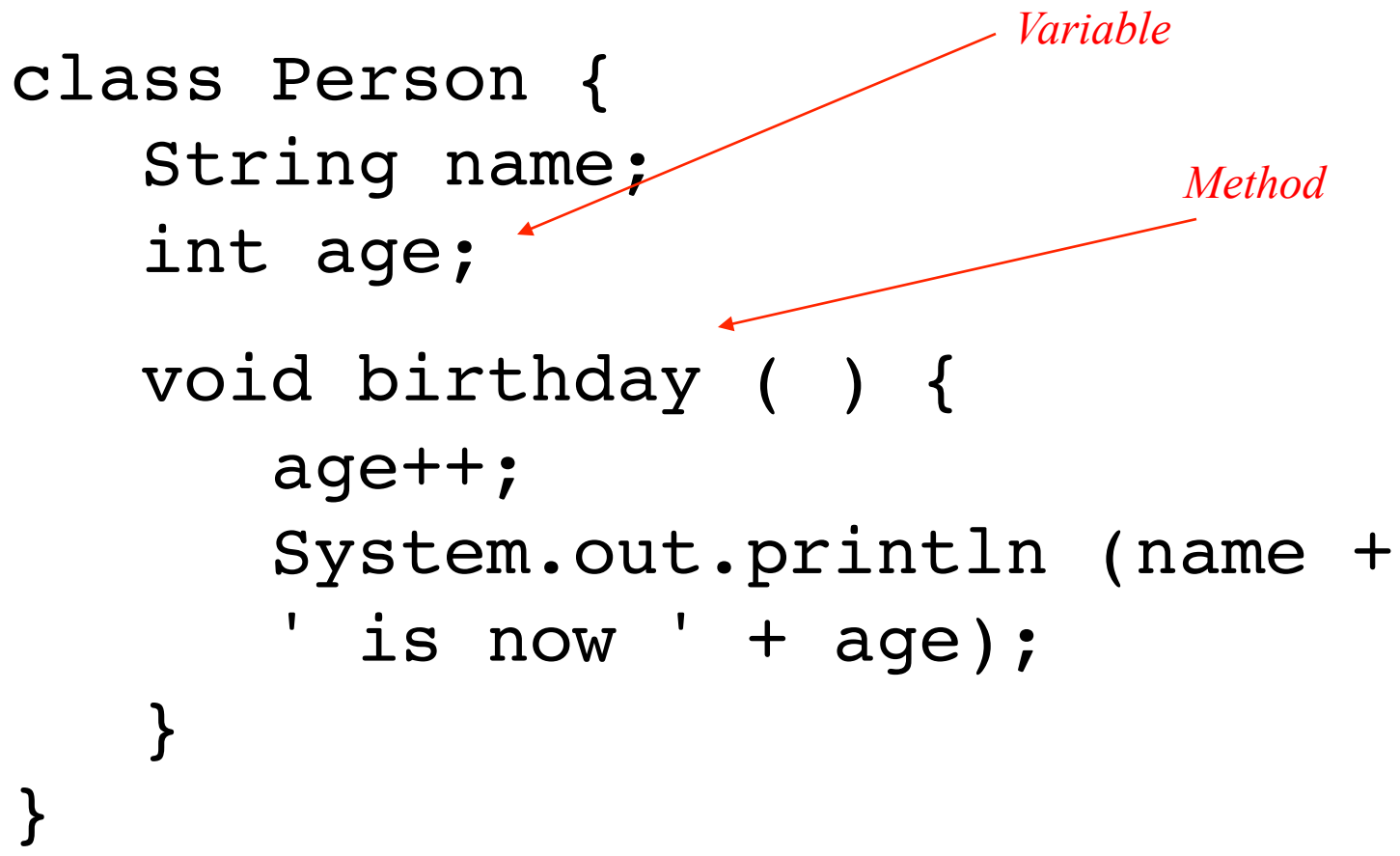
Objects

- identity – unique identification of an object
- attributes – data/state
- services – methods/operations
  - supported by the object
  - within objects responsibility to provide these services to other clients

Classes

- “type”
- object is an **instance** of class
- class groups similar objects
  - same (structure of) attributes
  - same services
- object holds values of its class’s attributes

An example of a class



Creation and Usage of objects

- Declaration - DataType identifier

```
Rectangle r1;
```

- Creation - new operator and specified constructor

```
r1 = new Rectangle();
Rectangle r2 = new Rectangle();
```

- Behavior - via the dot operator

```
r2.setSize(10, 20);
String s2 = r2.toString();
```

Interface

- A collection of abstract methods
- A class implements an interface, thereby inheriting the abstract methods of the interface.
- You cannot instantiate an interface.

Example

```
/* File name : Animal.java */
interface Animal {
        public void eat();
        public void travel();
}
```

Example(cont')

```java
/* File name : MammalInt.java */
public class MammalInt implements Animal{

   public void eat(){
      System.out.println("Mammal eats");
   }

   public void travel(){
      System.out.println("Mammal travels");
   }

   public int noOfLegs(){
      return 0;
   }

   public static void main(String args[]){
      MammalInt m = new MammalInt();
      m.eat();
      m.travel();
   }
}
```

OOD

- Java is OO language, provides features for OOD
  - Abstraction
  - Polymorphism
  - Inheritance
  - Encapusulation

Abstraction

- Provides a generalized view of your classes or object by providing relevant information.
- Focus on what the object does instead of how it does it.

Inheritance

- Class hierarchy
- Generalization and Specialization
  - subclass inherits attributes and services from its superclass
  - subclass may add new attributes and services
  - subclass may reuse the code in the superclass
  - subclasses provide specialized behaviors (overriding and dynamic binding)
  - partially define and implement common behaviors (abstract)

Example

```
abstract class Shape
{
        int  perimeter ();
}
```

Example

```
class Rectangle extends Shape{
    int  w, h;
    Rectangle (int ww, int hh) {
            w = ww;        h = hh;
    }
    int  perimeter () {
          return ( 2*(w + h) );
    }
}
```

Example(cont')

```
import java.awt.Color;
class  ColoredRectangle extends Rectangle {
  Color c;                              // inheritance
  ColoredRectangle (Color cc, int w, int h) {
            super(w,h);     c = cc; }
}

class  Square extends Rectangle {
  Square(int w) {
            super(w,w);  }
  int  perimeter () {          // overriding
            return ( 4*w );   }
}
```

Encapsulation

- Wrapping up data member and method together
- Hiding the internal details of an object

Example

```
public class EncapTest{
   private String name;

   public String getName(){
      return name;
   }

   public void setName(String newName){
      name = newName;
   }
}
```

Polymorphism

- Refers to the ability of an object to take on many forms.
- The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object.

Example

```
class Vehicle{
    public void move(){
    System.out.println(“Vehicles can move!!”);
    }
}

class MotorBike extends Vehicle{
    public void move(){
    System.out.println(“MotorBike can move and accelerate too!!”);
    }
}
```

Example

```
class Test{

    public static void main(String[] args){

    Vehicle vh=new MotorBike();
    vh.move();    // prints MotorBike can move and accelerate too!!

    vh=new Vehicle();
    vh.move();    // prints Vehicles can move!!

    }
}
```

Jar file

- Java archive (jar) files are compressed files that can store one or many files.
- Jar files normally contain java or class files, but other files may also be included.

Why use a JAR File?

- Compression:Jar files reduce the size of the original files.
- Speed: The applet can be downloaded in one http transaction.
- Security: Jar files can be signed digitally. Users who recognize the signature can optionally grant permission to run the file or refuse.
- Package sealing: Sealing a package within the jar file means that all classes defined in that package are found in the same jar file.

Create a jar file

- The jar command a utility that comes with the JDK.
- The format of the basic format of the jar command is:

```
Jar cf jar-file input-files
```

Create a jar file (cont')

- Take as an example the Java application made from the files
  - Foo.class
  - Foobar.class
- To make a jar file use the command

```
Jar cvf Foo.jar Foo.class Foobar.class
```

- Note that each file is separated by a space
- The execution of this command creates the file:
  - Foo.jar

Jar commands

- Here are the basic jar commands
  - View Jar file => `jar tf jar-file`
  - Extract Jar file => `jar xf jar-file`
- Jar applications can be run with the following basic command:
  - `java -jar jar-file`
- So to run our Foo.jar if it was an application use:
  - `java -jar Foo.jar`