IS311 Programming Concepts

JAVA

Interface & Polymorphism

ผศ. วันชัย ขันตี

ช่องทางสื่อสาร

- เบอร์โทรของผม 081 6434408 (วันชัย ขันตี)
- Home page
  http://www.bus.tu.ac.th/usr/wanchai/is311
- Facebook https://www.facebook.com/wanchaikhanti
- email: wanchai@tbs.tu.ac.th
- ติดต่อทางอีเมล์ทุกครั้งขอให้ใส่ subject ขึ้นต้นด้วย
  IS311-*xxxxxxxxxxx*
  ทุกครั้ง โดยที่ *xxxxxxxxxxx* หมายถึงเลขทะเบียนนักศึกษา
  ตัวอย่าง เช่น

  subject: IS311-5802123456 สอบถามปัญหาอินเทอร์เฟสครับ

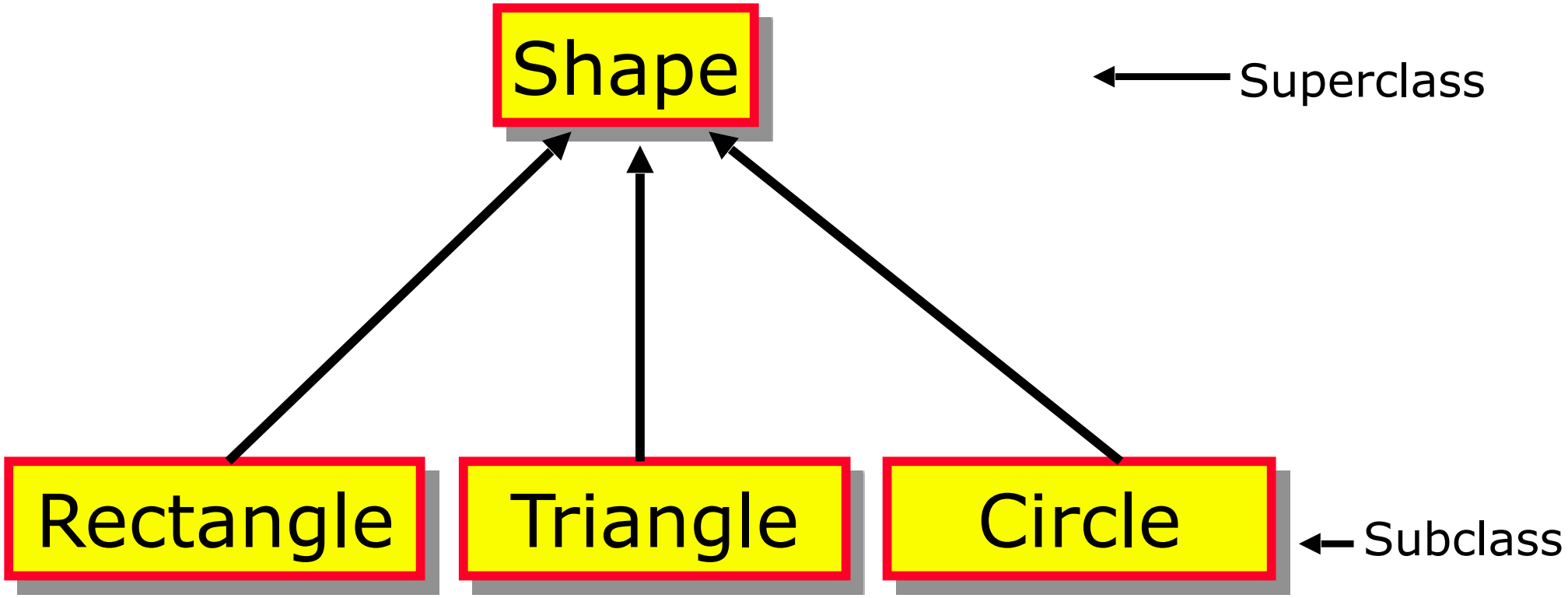
วัตถุประสงค์

- เพื่อให้รู้ความหมายและประโยชน์ของ interface
- เพื่อให้รู้วิธีเขียนและใช้ interface
- เพื่อให้รู้จักและรู้วิธีใช้ Polymorphism ขั้นพื้นฐาน

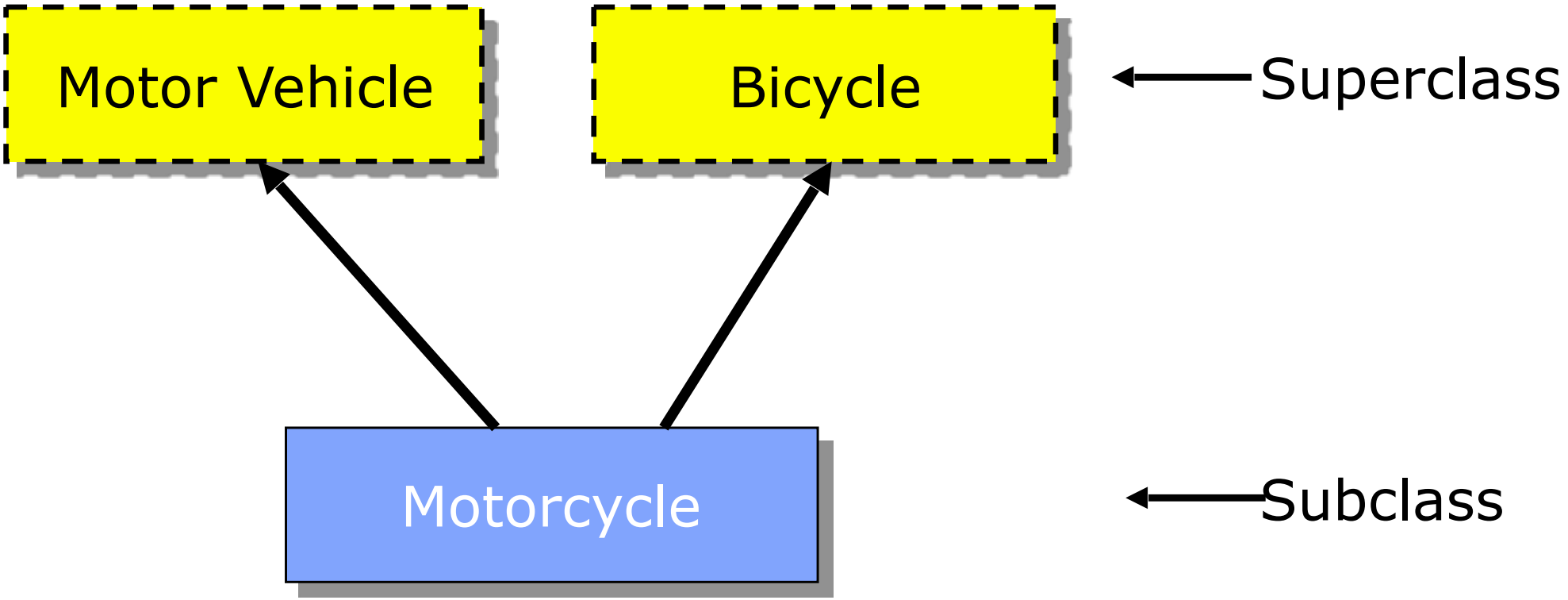
หัวข้อบรรยาย

- Multiple Inheritance
- ความหมายของ interface
- การกำหนด interface
- การใช้ interface
- ประโยชน์ของ interface
- เปรียบเทียบ interface กับ class
- polymorphism
- ตัวดำเนินการ instanceof

Single Inheritance



Multiple Inheritance



Interfaces

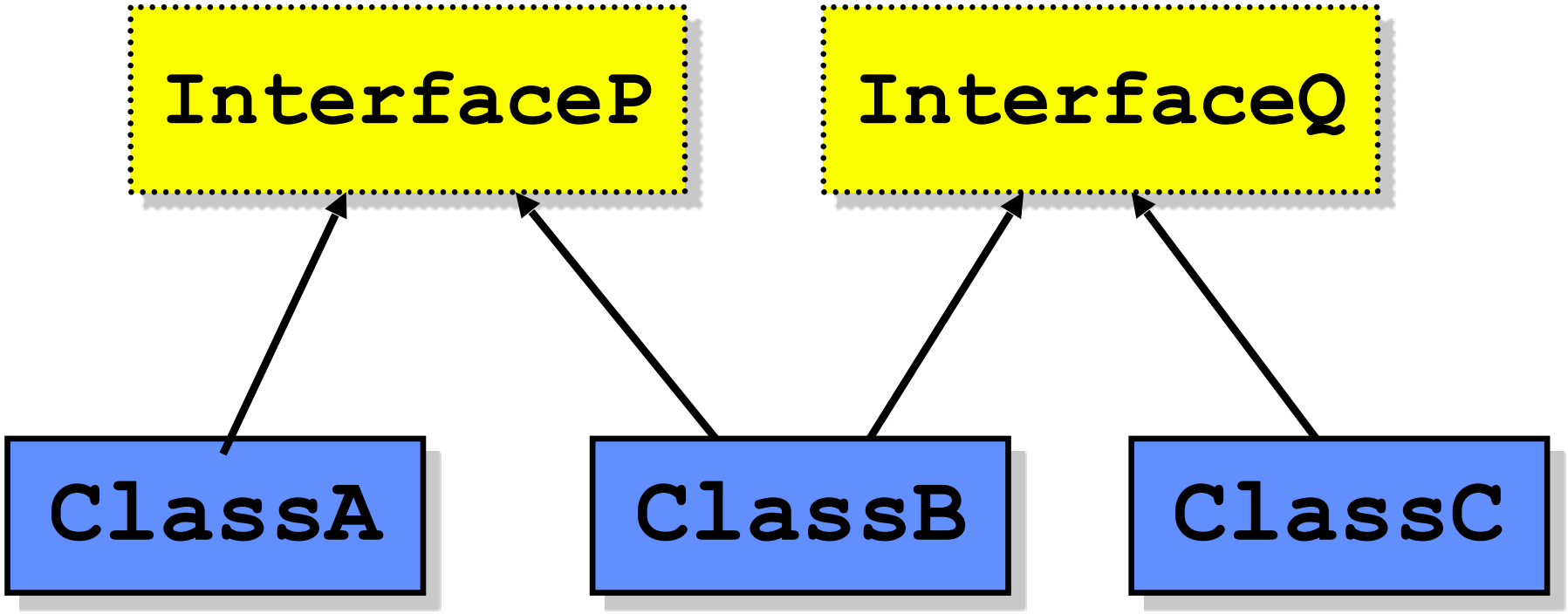
- Multiple inheritance considered too complex for Java
- Interfaces introduced to provide similar functionality
- Interfaces specify what methods a class supports
- Class may implement many interfaces

Interfaces (Cont.)

An *interface* defines a protocol of behavior that can be implemented by any class anywhere in the class hierarchy.

An interface defines a set of methods but does not implement them.

A class that implements the interface agrees to implement all the methods defined in the interface, thereby agreeing to certain behavior.



หนึ่งคลาสอาจอิมพลีเม้นต์ได้หลายอินเทอร์เฟส

หนึ่งอินเทอร์เฟสอาจถูกนำไปอิมพลีเม้นต์ได้ในหลายคลาส

Interface (cont.)

- An Interface is used to specify the form that something must have, but does not actually provide the implementation of that something
- An interface allows the programmer to describe a set of capabilities that a class must implement.

Interface is a Special Class

An interface is treated like a special class in Java. Each interface is compiled into a separate bytecode file, just like a regular class. Like an abstract class, you cannot create an instance from an interface using the new operator, but in most cases you can use an interface more or less the same way you use an abstract class. For example, you can use an interface as a data type for a variable, as the result of casting, and so on.

The `interface` Statement

```
modifier interface InterfaceName {
  return-type methodName(parameter-list);
  type final-varname = value;
}
```

- Similar to a class definition
- *InterfaceName* เป็นชื่ออินเทอร์เฟส ตั้งตามกฎการตั้งชื่อ identifier
- All methods are implicitly `public` and `abstract`
- All variables are implicitly `public`, `static` and `final`

ตัวอย่างการเขียน interface

Interface declaration →

Interface body →

```
public interface Comparable
{
    int compareTo(Object o);
}
```

```
public interface Printable {
    void print();
}
```

method declaration มีแต่ signature ไม่มี body ของเมท็อด

```
public interface Thammasat {
  int NUMBER_OF_CAMPUSES=2;
  int NUMBER_OF_FACULTIES=16;
}
```

Constant declaration ประกาศตัวแปรแล้วเปลี่ยนค่าภายหลังไม่ได้ จึงต้องกำหนดค่าตั้งแต่ต้น

The **implements** Statement

A class inherits an interface using the ***implements*** keyword.

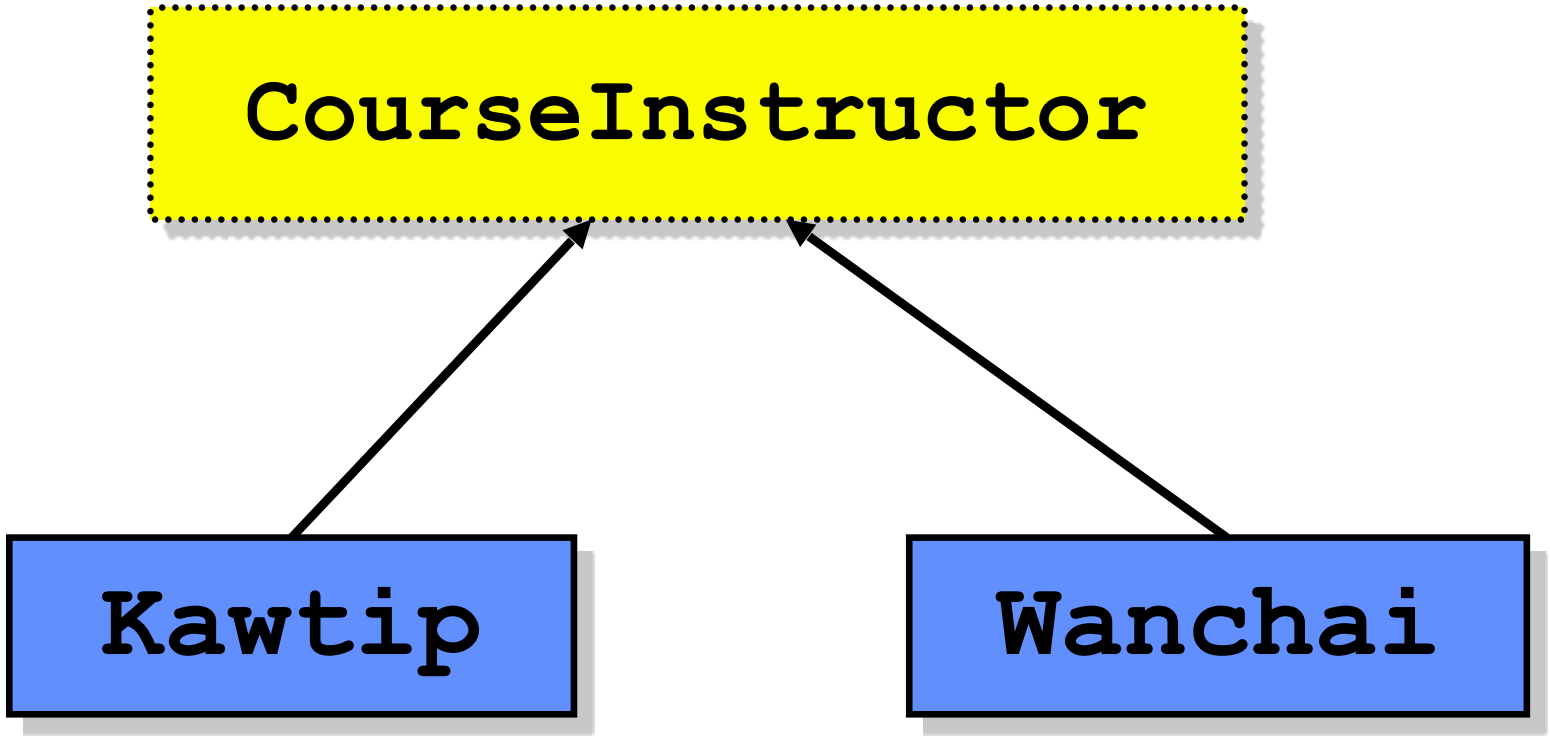
```
[modifier] class classname
     [extends superclassname]
  [implements interface1
                    [,interface2]...]
{
     [variable declaration;]
     [method declaration;]
}
```

ตัวอย่างการใช้ interface

```
public interface Printable {
    void print();
}
```

```
class SavingAccount extends Account
                    implements Printable
{
  public void print() {
    System.out.println("Account Number: " + no);
    System.out.println("Account Type: Saving");
    System.out.println("Balance: " + balance);
   }
}
```

หลายคลาสใช้อินเทอร์เฟสเดียวกัน



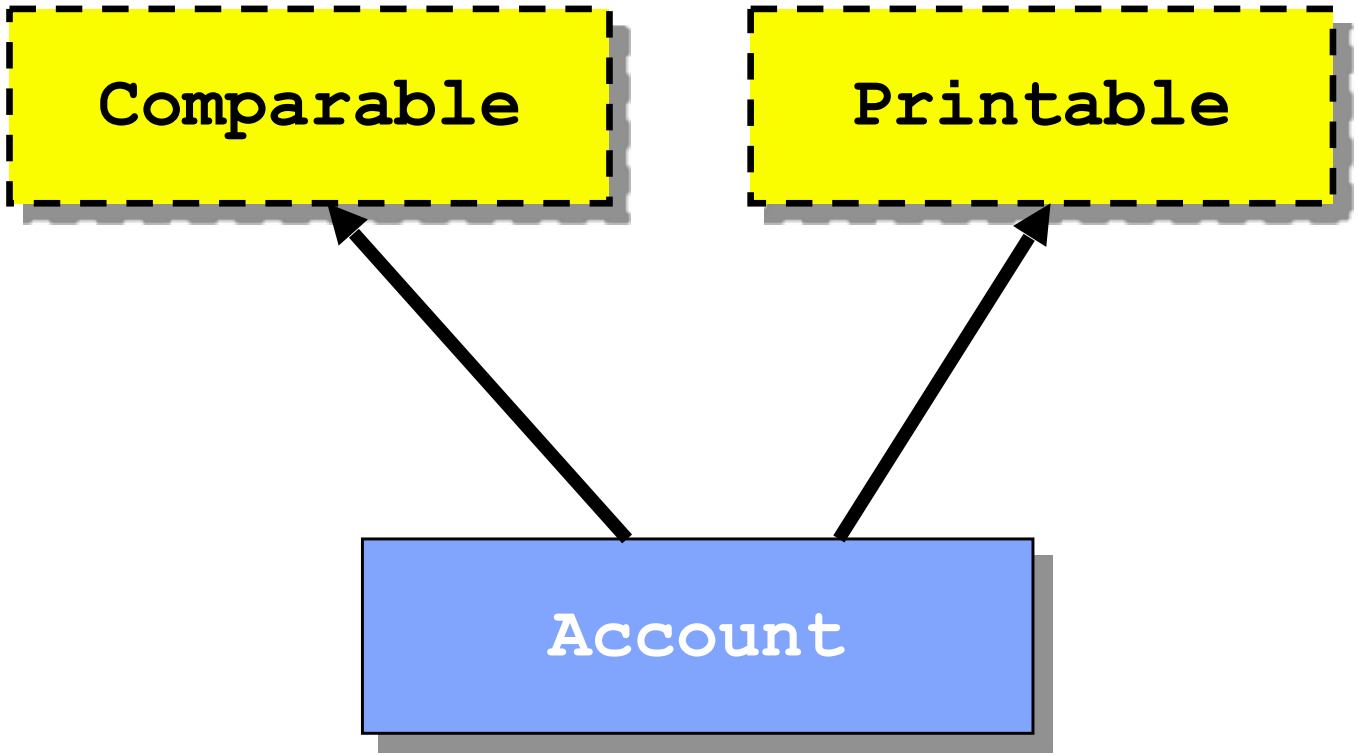
ถึงจะ implement อินเทอร์เฟสเดียวกัน ไม่ได้ทำให้คลาส **Kawtip** และ **Wanchai** มีความสัมพันธ์กัน เพียงแต่มีอะไรบางอย่าง (ตามที่กำหนดไว้ในอินเทอร์เฟส) ที่เหมือนกันเท่านั้น

Interface Example


```
interface CourseInstructor{
  void printInstructorName();
}
class Kawtip implements CourseInstructor{
...
  public void printInstructorName(){
      System.out.println("Kawtip");
  }
}
class Wanchai implements CourseInstructor{
...
  public void printInstructorName(){
      System.out.println("Wanchai");
  }
}
```

ทั้งคลาส **Kawtip** และ **Wanchai** ต่างก็มีเมท็อด **printInstructorName** ให้ใช้ทั้งคู่ โดยที่ 2 คลาสนี้ไม่ได้มีความสัมพันธ์แบบ inheritance กันเลย

หนึ่งคลาสใช้หลายอินเทอร์เฟส



ตัวอย่าง 1 คลาส 2 อินเทอร์เฟส

```
class SavingAccount extends Account
                    implements Printable,
                               Comparable
{
  public void print() {
    System.out.println("Account Number: " + no);
    System.out.println("Balance: " + balance);
   }
  public int compareTo(Object obj) {
     Account ac = (Account) obj;
     return balance - ac.balance;
  }
}
```

ตัวอย่างการใช้อินเทอร์เฟสระบุค่าคงที่

```
interface InterestRate{
   float FIXED12MONTHS = 2.55f;
   float FIXED6MONTHS  = 2.25f;
   float FIXED3MONTHS  = 1.75f;
}
class Fixed3Account implements InterestRate {
...
   public void computeInterest(){
      interest = principle * FIXED3MONTHS * 3.0 / 12 / 100 );
   }
}
class Fixed6Account implements InterestRate {
...
   public void computeInterest(){
      interest = principle * FIXED6MONTHS * 6.0 / 12 / 100 );
   }
}
```

Using an Interface as a Type

- อินเทอร์เฟสสามารถนำมาใช้เป็นชนิดของข้อมูล (data type) ได้ เช่นเดียวกับคลาส (เป็น reference type)
- เราสามารถใช้ตัวแปรอ้างอิงที่มีชนิดเป็นอินเทอร์เฟสในการอ้างถึงอ็อบเจกต์ที่สร้างมาจากคลาสที่อิมพลีเม้นต์อินเทอร์เฟสนั้นได้ (อย่าลืมว่าอินเทอร์เฟสนำไปสร้างอ๊อบเจกต์ไม่ได้ แต่คลาสที่อิมพลีเม้นต์ทุก abstract method อย่างครบถ้วนแล้วสามารถสร้างอ็อบเจกต์ได้)
- reference type ที่เป็นอินเทอร์เฟส (ซึ่งนำไปอ้างถึงอ็อบเจกต์)สามารถอ้างถึงได้เฉพาะสมาชิกที่กำหนดไว้ในอินเทอร์เฟสนั้น ๆ เท่านั้น

Interface are used to express the commonality between classes

การใช้อ็อบเจ็กต์ที่มีการ implement อินเทอร์เฟส

```
Kawtip kaw = new Kawtip();
kaw.printInstructorName();

Wanchai chai = new Wanchai();
chai.printInstructorName();
```

การใช้ตัวแปรอ้างอิงชนิด `interface` ในการอ้างถึงอ็อบเจกต์

```
CourseInstructor  ci;
ci = kaw;
ci.printInstructorName();
ci = chai;
ci.printInstructorName();
```

Advantages of Interface

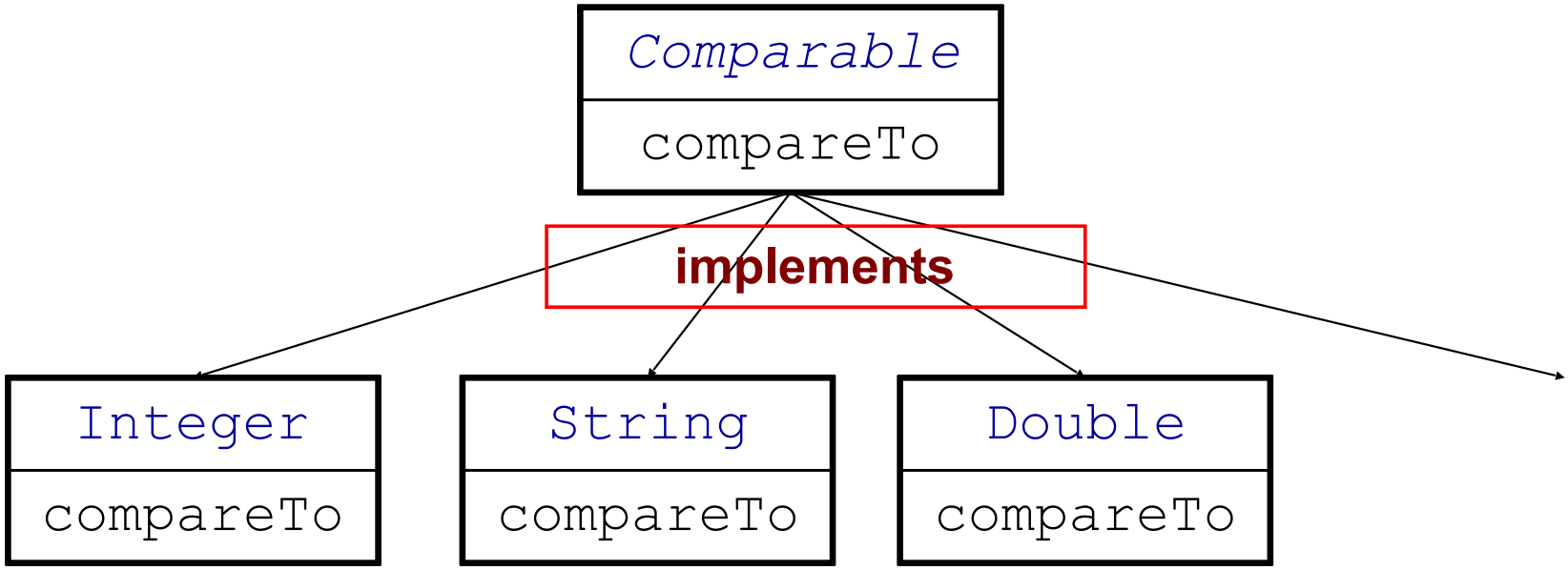
- capturing similarities between unrelated classes without forcing a class relationship.
- declaring methods that one or more classes are expected to implement.
- revealing an object's programming interface without revealing its class (objects such as these are called anonymous objects and can be useful when shipping a package of classes to other developers)

Interface Modifiers

Modifier	Description
<blank>	interface is accessible to objects within the current package
public	interface is accessible to objects outside the package it is part of. Only one public interface is permitted per source code file.
protected	
private	

Java Interface - *Comparable*

เป็นอินเทอร์เฟสมาตรฐานของจาวาที่ใช้บ่อย ถูกนำไป implement ในคลาสจำนวนมาก




```
int compareTo(Object o)
```

Compares this object with the specified object for order.
Returns a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object.


Comparable (Java Platform SE 6) - Windows Intern
D:\Java6\docs\api\index.html
File Edit View Favorites Tools Help
Comparable (Java Platform SE 6)
อ่าน online document ที่
http://download.oracle.com/javase/6/docs/api//
Comparable เป็นอินเทอร์เฟสอยู่ใน
แพคเกจ java.lang
java.awt.geom
java.awt.im
java.awt.im.spi
java.awt.image
java.awt.image.renderabl
java.awt.print
java.beans
java.beans.beancontext
java.io
java.lang
java.lang.annotation
java.lang.instrument
java.lang
Interfaces
Appendable
CharSequence
Cloneable
Comparable
Iterable
Readable
Runnable
Thread.UncaughtExceptio
Classes
Boolean
Byte
Character
Character.Subset
Character.UnicodeBlock
ชื่ออินเทอร์เฟส แสดงด้วยตัวเอียง
java.lang
Interface Comparable<T>
Type Parameters:
T - the type of objects that this object may be compared to
All Known Subinterfaces:
Delayed, Name, RunnableScheduledFuture<V>, ScheduledFuture<V>
All Known Implementing Classes:
Authenticator.RequestorType, BigDecimal, BigInteger, Boolean, Byte, ByteBuffer, Calendar,
Character, CharBuffer, Charset, ClientInfoStatus, CollationKey,
Component.BaselineResizeBehavior, CompositeName, CompoundName, Date, Date,
Desktop.Action, Diagnostic.Kind, Dialog.ModalExclusionType, Dialog.ModalityType, Double,
DoubleBuffer, DropMode, ElementKind, ElementType, Enum, File, Float, FloatBuffer,
Formatter.BigDecimalLayoutForm, FormSubmitEvent.MethodType, GregorianCalendar,
GroupLayout.Alignment, IntBuffer, Integer, JavaFileObject.Kind, JTable.PrintMode,
KeyRep.Type, LayoutStyle.ComponentPlacement, LdapName, Long, LongBuffer,
MessageContext.Scope, Modifier,
MultipleGradientPaint.CycleMethod, NestingKind,
ObjectStreamField, Proxy.Type, Rdn,
Resource.AuthenticationType, RetentionPolicy, RoundingMode, RowFilter.ComparisonType,
RowIdLifetime, RowSorterEvent.Type, Service.Mode, Short, ShortBuffer,
SOAPBinding.ParameterStyle, SOAPBinding.Style, SOAPBinding.Use, SortOrder, SourceVersion,
SSLEngineResult.HandshakeStatus, SSLEngineResult.Status, StandardLocation, String,
SwingWorker.StateValue, Thread.State, Time, Timestamp, TimeUnit, TrayIcon.MessageType,
TypeKind, URI, UUID, WebParam.Mode, XmlAccessOrder, XmlAccessType, XmlNsForm
My Computer
100%

The `instanceof` operator

ตัวอย่าง

```
if ( anObject instanceof aClass) {
   ...
}
```

เมื่อ *anObject* เป็นตัวแปรหรือค่าอ้างอิง ส่วน *aClass* เป็นชื่อคลาสหรือชื่ออินเทอร์เฟส

ตัวดำเนินการ **`instanceof`** ให้ผลลัพธ์เป็น `boolean true` ถ้า `aObject` เป็นตัวอย่างหนึ่ง(instance) ของ `aClass` ถ้าไม่ใช่ให้จะให้ผลลัพธ์เป็น `false`

`String` and `Date` Classes

Many classes (e.g., **`String`** and **`Date`**) in the Java library implement **`Comparable`** to define a natural order for the objects. If you examine the source code of these classes, you will see the keyword implements used in the classes, as shown below:

```
public class String extends Object
    implements Comparable {
  // class body omitted
}
```

```
public class Date extends Object
    implements Comparable {
  // class body omitted
}
```

```
new String() instanceof String
new String() instanceof Comparable
new java.util.Date() instanceof java.util.Date
new java.util.Date() instanceof Comparable
```

ข้อแตกต่างระหว่าง Interface และ Class

- An interface cannot implement any methods, whereas an abstract class can.
- A class can implement many interfaces but can have only one superclass.
- An interface is not part of the class hierarchy. Unrelated classes can implement the same interface.

Interfaces vs. Abstract Classes

In an interface, the data must be constants; an abstract class can have all types of data.

Each method in an interface has only a signature without implementation; an abstract class can have concrete methods.

	Variables	Constructors	Methods
Abstract class	No restrictions	Constructors are invoked by subclasses through constructor chaining. An abstract class cannot be instantiated using the new operator.	No restrictions.
Interface	All variables must be public static final	No constructors. An interface cannot be instantiated using the new operator.	All methods must be public abstract instance methods

Interfaces vs. Abstract Classes, cont.

All data fields are *public final static* and all methods are *public abstract* in an interface. For this reason, these modifiers can be omitted, as shown below:

```
public interface T1 {
  public static final int K = 1;

  public abstract void p();
}
```

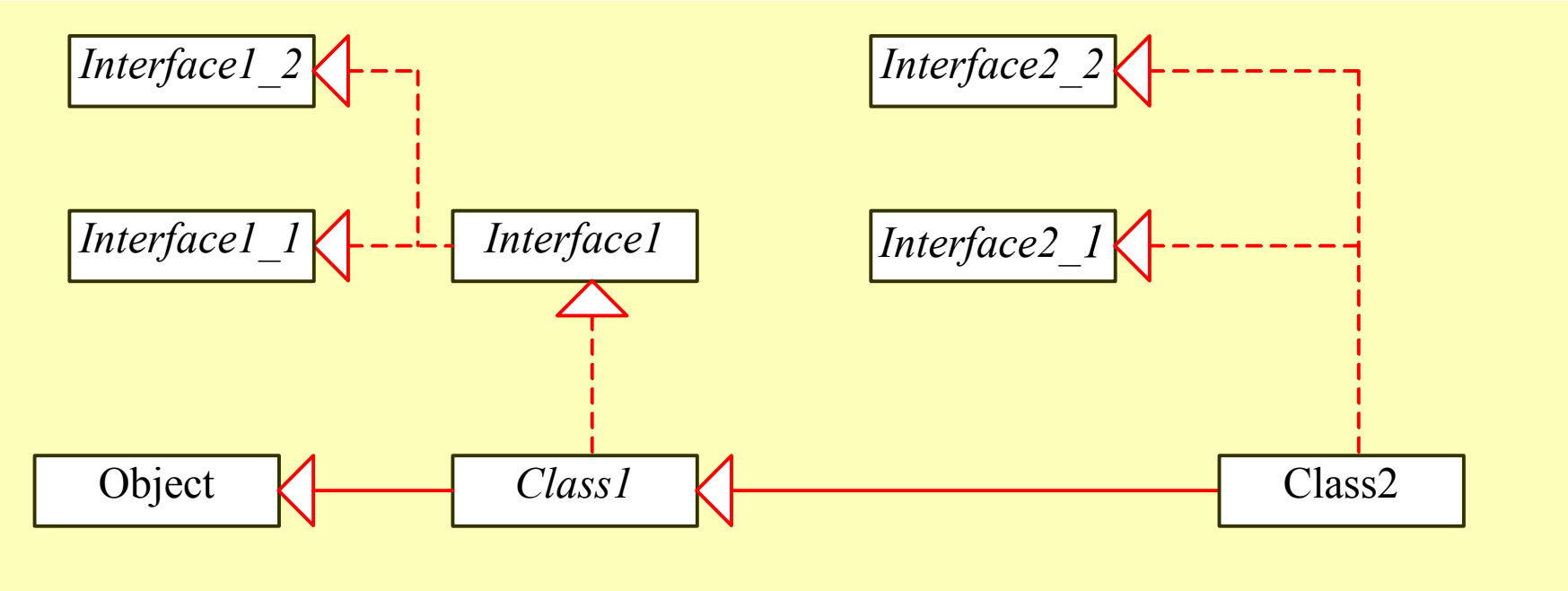
Equivalent

```
public interface T1 {
  int K = 1;

  void p();
}
```

A constant defined in an interface can be accessed using syntax `InterfaceName.CONSTANT_NAME` (e.g., `T1.K`)

Interfaces vs. Abstract Classes, cont.

All classes share a single root, the **`Object`** class, but there is no single root for interfaces. Like a class, an interface also defines a type. A variable of an interface type can reference any instance of the class that implements the interface. If a class extends an interface, this interface plays the same role as a superclass. You can use an interface as a data type and cast a variable of an interface type to its subclass, and vice versa.



Suppose that `c` is an instance of `Class2`. `c` is also an instance of **`Object`**, `Class1`, `Interface1`, `Interface1_1`, `Interface1_2`, `Interface2_1`, and `Interface2_2`.

Caution: conflict interfaces

In rare occasions, a class may implement two interfaces with conflict information (e.g., two same constants with different values or two methods with same signature but different return type). This type of errors will be detected by the compiler.

Whether to use an interface or a class?

Abstract classes and interfaces can both be used to model common features. How do you decide whether to use an interface or a class? In general, a strong is-a relationship that clearly describes a parent-child relationship should be modeled using classes. For example, a staff member is a person. So their relationship should be modeled using class inheritance. A weak is-a relationship, also known as an is-kind-of relationship, indicates that an object possesses a certain property. A weak is-a relationship can be modeled using interfaces. For example, all strings are comparable, so the String class implements the Comparable interface. You can also use interfaces to circumvent single inheritance restriction if multiple inheritance is desired. In the case of multiple inheritance, you have to design one as a superclass, and others as interface.

Creating Custom Interfaces


```
public interface Edible {
  /** Describe how to eat */
  public String howToEat();
}
```

```
class Animal {
}

class Chicken extends Animal
    implements Edible {
  public String howToEat() {
    return "Fry it";
  }
}

class Tiger extends Animal {
}
```

```
class abstract Fruit
    implements Edible {
}

class Apple extends Fruit {
  public String howToEat() {
    return "Make apple cider";
  }
}

class Orange extends Fruit {
  public String howToEat() {
    return "Make orange juice";
  }
}
```

Implements Multiple Interfaces


```
class Chicken extends Animal implements
                        Edible, Comparable {
  int weight;
  public Chicken(int weight) {
    this.weight = weight;
  }
  public String howToEat() {
    return "Fry it";
  }
  public int compareTo(Object o) {
    return weight - ((Chicken)o).weight;
  }
}
```

Creating Custom Interfaces, cont.


```
public interface Edible {
  /** Describe how to eat */
  public String howToEat();
}
```

```
public class TestEdible {
  public static void main(String[] args) {
    Object[] objects = {new Tiger(), new Chicken(), new Apple()};
    for (int i = 0; i < objects.length; i++)
      showObject(objects[i]);
  }

  public static void showObject(Object object) {
    if (object instanceof Edible)
      System.out.println(((Edible)object).howToEat());
  }
}
```

The `Cloneable` Interfaces

Marker Interface: An empty interface.

A marker interface does not contain constants or methods. It is used to denote that a class possesses certain desirable properties. A class that implements the `Cloneable` interface is marked cloneable, and its objects can be cloned using the `clone()` method defined in the `Object` class.

```
package java.lang;
public interface Cloneable {
}
```

Examples

Many classes (e.g., **`Date`** and **`Calendar`**) in the Java library implement ***`Cloneable`***. Thus, the instances of these classes can be cloned. For example, the following code

```
Calendar calendar = new GregorianCalendar(2003, 2, 1);
Calendar calendarCopy = (Calendar)calendar.clone();
System.out.println("calendar == calendarCopy is " +
  (calendar == calendarCopy));
System.out.println("calendar.equals(calendarCopy) is " +
  calendar.equals(calendarCopy));
```

displays

```
calendar == calendarCopy is false
calendar.equals(calendarCopy) is true
```

Assignment
Implement Your Own ***Comparable***

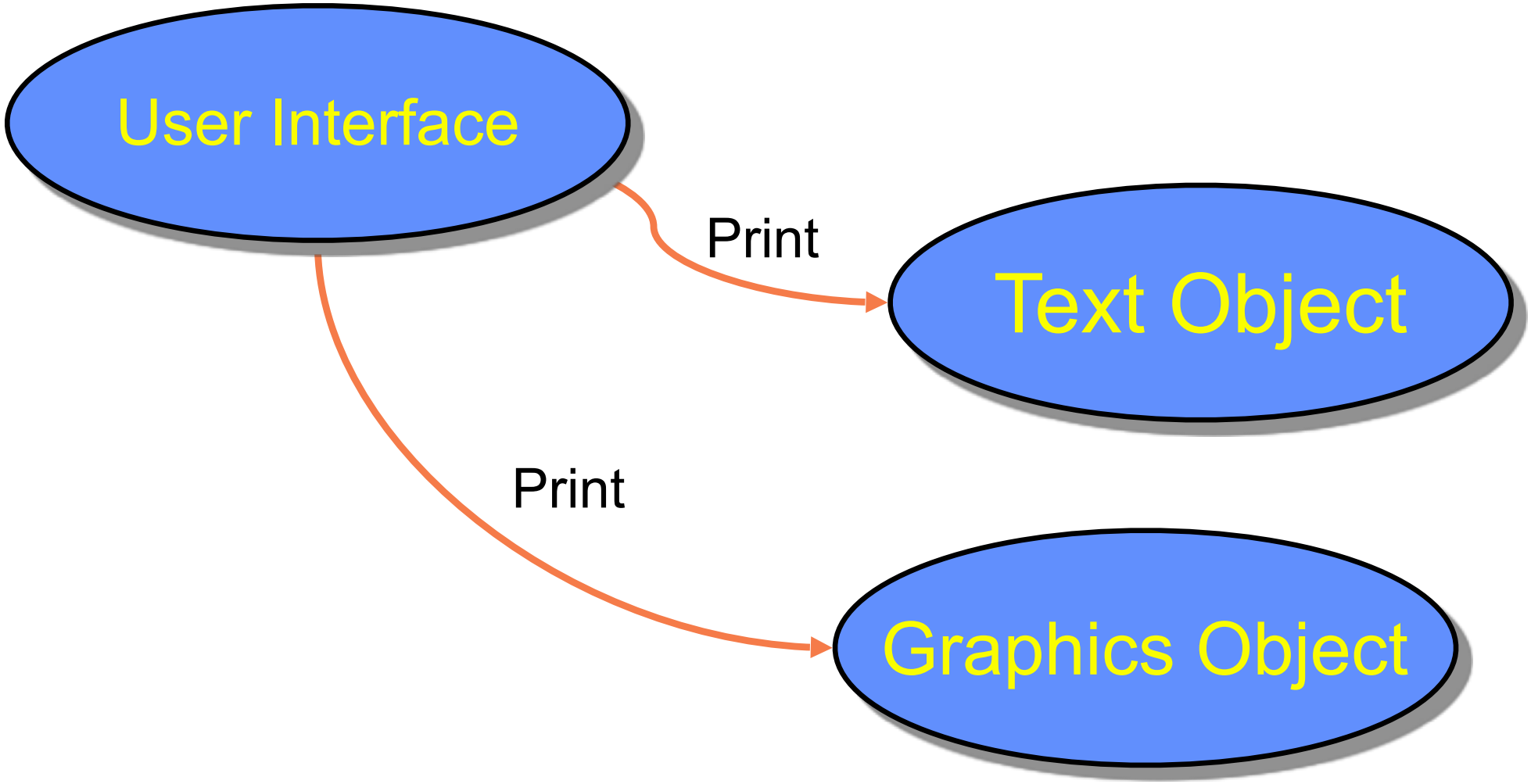
Comparable ratios

```
public class Ratio implements Comparable
{
    protected int numerator;
    protected int denominator;
    public Ratio (int top, int bottom)  //pre: bottom !=0
    { numerator = top; denominator = bottom; }
    public int getNumerator() { return numerator; }
    public int getDenominator() { return denominator; }

    public int compareTo(Object other)//pre: other is non-
                                      //null Ratio object
    { /* Your code  */ }
}
```

ดาวน์โหลด source code **ในเว็บไซต์ไปเขียนเพิ่ม**

ภาวะพหุสัณฐาน(Polymorphism)

Polymorphism



```
class Holiday {
  void celebrate() {
    System.out.println("It\'s Holiday");
  }
}

class Christmas extends Holiday {
  void celebrate() {
    System.out.println("It\'s Christmas");
  }
}

class Test {
  public static void main(String []args) {
    Holiday day = new Christmas();
    day.celebrate();
  }
}
```

โปรแกรมนี้จะพิมพ์อะไร

Polymorphism

- A polymorphic reference is one which can refer to one of several possible methods
- Suppose the **`Holiday`** class has a method called **`celebrate`**, and the **`Christmas`** class overrode it
- Now consider the following invocation:

```
day.celebrate();
```

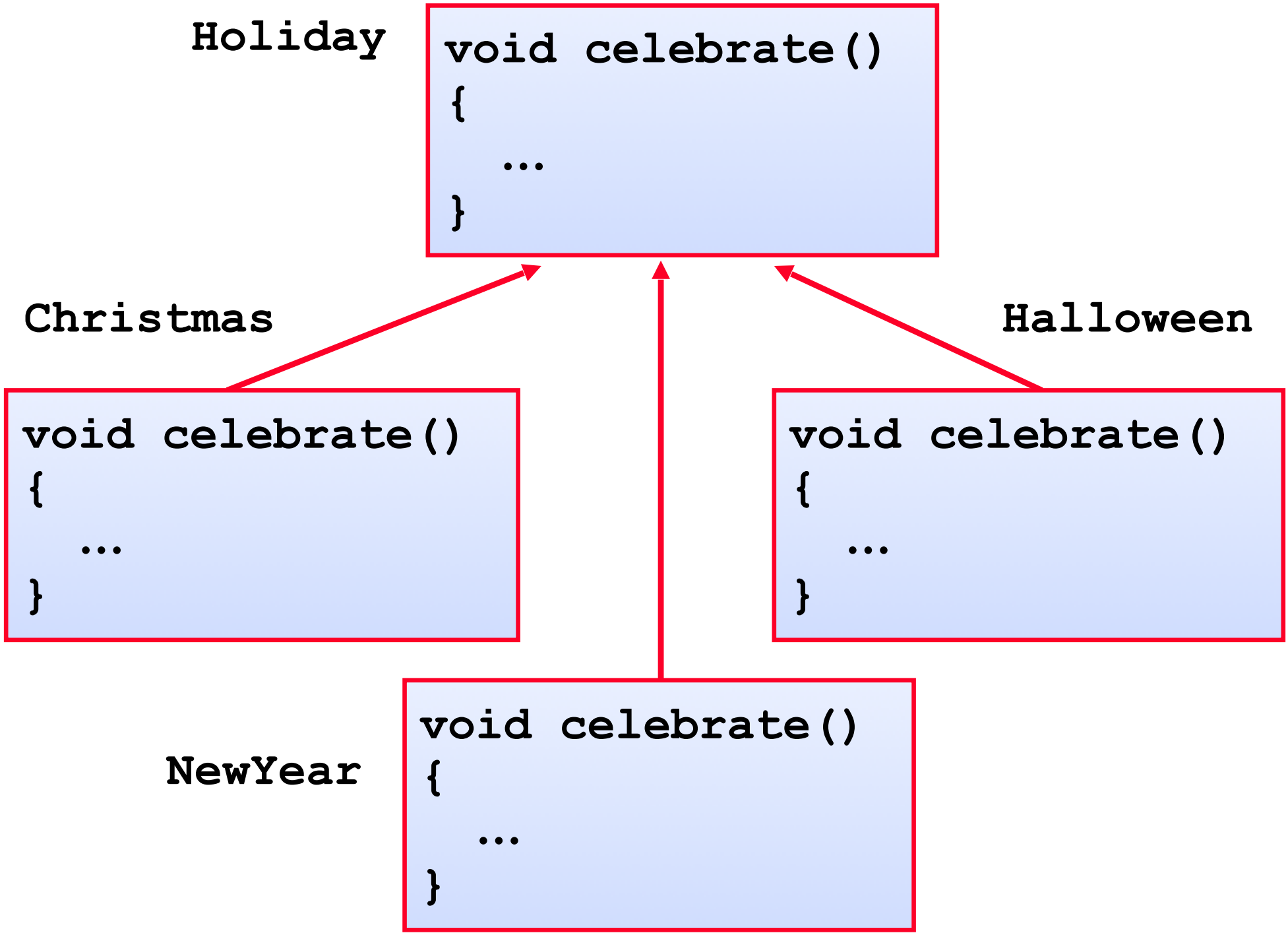
- If `day` refers to a **`Holiday`** object, it invokes Holiday's version of celebrate; if it refers to a **`Christmas`** object, it invokes that version

```
class Holiday {
  void celebrate() {
    System.out.println("It\'s Holiday");
  }
}

class Christmas extends Holiday {
}

class Test {
  public static void main(String args[]) {
    Holiday day = new Christmas();
    day.celebrate();
  }
}
```

โปรแกรมนี้มี Syntax Error หรือไม่ ถ้าไม่ จะพิมพ์อะไร

Polymorphism

- In general, it is the type of the object being referenced, not the reference type, that determines which method is invoked
- Note that, if an invocation is in a loop, the exact same line of code could execute different methods at different times
- Polymorphic references are therefore resolved at run-time, not during compilation (เรียกว่า late binding)


Holiday
void celebrate()
{
…
}
Christmas
void celebrate()
{
…
}
Halloween
void celebrate()
{
…
}
NewYear
void celebrate()
{
…
}

```
void m(Holiday  day) {
  day.celebrate();
  // day is polymorphic reference
}
…

Christmas xmas = new Christmas();
m(xmas);
m(new Halloween());
m(new NewYear());
```

Polymorphism

- Note that, because all classes inherit from the `Object` class, an `Object` reference can refer to any type of object
- เราสามารถใช้ตัวดำเนินการ `instanceof` ในการตรวจสอบว่าอ๊อบเจกต์ที่เราสนใจเป็นอ๊อบเจกต์ของคลาสที่เราสงสัยหรือไม่ หรือพูดอีกนัยหนึ่งคือมี type เป็นคลาสหรืออินเทอร์เฟสที่เราสงสัยหรือเปล่า

```
void m(Holiday  day) {
   day.celebrate();
   if (day instanceof Christmas)
     System.out.println("Hello Santa");
   if (day instanceof Halloween)
     System.out.println("Hello Satan");
}
```

Polymorphism

- An object's ability to decide what method to apply to itself, depending on where it is in the inheritance hierarchy, is usually called *polymorphism*. The idea behind polymorphism is that the message may be the same, objects may respond differently.
- Polymorphism can apply to any method that is inherited from a superclass.
- Polymorphism simply means that you can use any class in place of its parent class.

Interface & Polymorphism

When multiple classes implement the same interface, each class implements the methods of the interface in different ways.

```
public interface Printable {
    void print();
}
class Account
{
    int no;
    double balance;
}

class SavingAccount extends Account
                    implements Printable
{
    public void print() {
        System.out.println("Account Number: " + no);
        System.out.println("Account Type: Saving");
        System.out.println("Balance: " + balance);
    }
}
```

```
class Wanchai implements Printable {
      public void print(){
            System.out.println("Name: Wanchai");
      }
}
class Test1 {
      public static void main(String [] args) {
            SavingAccount sa = new SavingAccount();
            Wanchai wa = new Wanchai();
            sa.print();
            wa.print();
      }
}
class Test2 {
      public static void main(String [] args) {
            Printable sa = new SavingAccount();
            Printable wa = new Wanchai();
            sa.print();
            wa.print();
      }
}
```

```
class Test3 {
  public static void main(String []args) {
    Printable [] pa = { new SavingAccount(), new Wanchai()};
      for (int i=0; i < pa.length; i++)
        pa[i].print();
  }
}
class Test4 {
  public static void main(String []args) {
    Object [] pa = { new SavingAccount(), new Wanchai()};
      for (int i=0; i < pa.length; i++)
        ((Printable)pa[i]).print();
  }
}
class Test5 {
  public static void main(String []args) {
    Object [] pa =
      { new SavingAccount(), new Wanchai(), "Hello"};
      for (int i=0; i < pa.length; i++) {
        if (pa[i] instanceof Printable)
          ((Printable)pa[i]).print();
      }
  }   // หมายเหตุ สีแดงหมายถึงส่วนที่เขียนเพิ่มเติมหรือเปลี่ยนไปจากคลาสตัวอย่างก่อนหน้านี้
}
```

Advantages of polymorphism

- Two or more object classes are able to respond differently to one message.
- An object does not need to know to whom it is sending a message.
- A method name can be implemented in different ways.