OBJECTIVES

封藏性

- In this chapter you will learn:
- Encapsulation and data hiding.
- The notions of data abstraction and abstract data types (ADTs).
- **`Constructor`** and **`Finalize`**
- To use keyword **`this`**.
- To use **`static`** variables and methods.
- To import `static` members of a class.
- To use the `enum` type to create sets of constants with unique identifiers.

New

- How to declare `enum` constants with parameters.
- To create **`package`** and the way to import package.
- package access

Outline

Time1.java

(1 of 2)

```
// Fig. 8.1: Time1.java
// Time1 class declaration maintains the time in 24-hour format.

public class Time1
{
   private int hour;   // 0 – 23
   private int minute; // 0 – 59
   private int second; // 0 – 59

   // set a new time value using universal time; ensure that
   // the data remains consistent by setting invalid values to zero
   public void setTime( int h, int m, int s )

      hour = ( ( h >= 0 && h < 24 ) ? h : 0 );   // validate hour
      minute = ( ( m >= 0 && m < 60 ) ? m : 0 ); // validate minute
      second = ( ( s >= 0 && s < 60 ) ? s : 0 ); // validate second
   } // end method setTime

```

private instance variables

Declare **public** method **setTime**

Validate parameter values before setting instance variables

養成好習慣：

先驗證正確性再修改內容

Outline

Time1.java

(2 of 2)

```
   // convert to String in universal-time format (HH:MM:SS)
   public String toUniversalString()
   {
      return String.format( "%02d:%02d:%02d", hour, minute, second );
   } // end method toUniversalString

   // convert to String in standard-time format (H:MM:SS AM or PM)
   public String toString()
   {
      return String.format( "%d:%02d:%02d %s",
         ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 ),
         minute, second, ( hour < 12 ? "AM" : "PM" ) );
   } // end method toString
} // end class Time1
```

format strings

Time Class 只是一個定義而已，祇是一個物件的『模具』而已

Time1
private int hour private int minute private int second
public void setTime (int, int, int) public String toUniversalString() public String toString()

Outline

Time1Test.java

(1 of 2)

```
// Fig. 8.2: Time1Test.java
// Time1 object used in an application.

public class Time1Test
{
   public static void main( String args[] )
   {
      // create and initialize a Time1 object
      Time1 time = new Time1(); // invokes Time1 constructor

      // output string representations of the time
      System.out.print( "The initial universal time is: " );
      System.out.println( time.toUniversalString() );
      System.out.print( "The initial standard time is: " );
      System.out.println( time.toString() );
      System.out.println(); // output a blank line

```

Create a **Time1** object

Call **toUniversalString** method

Call **toString** method

Outline

Time1Test.java

```
      // change time and output updated time
      time.setTime( 13, 27, 6 );
      System.out.print( "Universal time after setTime is: " );
      System.out.println( time.toUniversalString() );
      System.out.print( "Standard time after setTime is: " );
      System.out.println( time.toString() );
      System.out.println(); // output a blank line

      // set time with invalid values; output updated time
      time.setTime( 99, 99, 99 );
      System.out.println( "After attempting invalid settings:" );
      System.out.print( "Universal time: " );
      System.out.println( time.toUniversalString() );
      System.out.print( "Standard time: " );
      System.out.println( time.toString() );
   } // end main
} // end class Time1Test
```

Call **setTime** method

Call **setTime** method with invalid values

先驗證參數正確性的重要性！

```
The initial universal time is: 00:00:00
The initial standard time is: 12:00:00 AM

Universal time after setTime is: 13:27:06
Standard time after setTime is: 1:27:06 PM

After attempting invalid settings:
Universal time: 00:00:00
Standard time: 12:00:00 AM
```

8.3 Controlling Access to Members

- A class's public interface
  - `public` methods a view of the services the class provides to the class's clients
- A class's implementation details
  - `private` variables and `private` methods are not accessible to the class's clients

Outline

MemberAccessTest.java

```
// Fig. 8.3: MemberAccessTest.java
// Private members of class Time1 are not accessible.
public class MemberAccessTest
{
   public static void main( String args[] )
   {
      Time1 time = new Time1(); // create and initialize Time1 object

      time.hour = 7;     // error: hour has private access in Time1
      time.minute = 15; // error: minute has private access in Time1
      time.second = 30; // error: second has private access in Time1
   } // end main
} // end class MemberAccessTest
```

Attempting to access **private** instance variables

```
MemberAccessTest.java:9: hour has private access in Time1
      time.hour = 7;     // error: hour has private access in Time1
          ^
MemberAccessTest.java:10: minute has private access in Time1
      time.minute = 15; // error: minute has private access in Time1
          ^
MemberAccessTest.java:11: second has private access in Time1
      time.second = 30; // error: second has private access in Time1
          ^
3 errors
```

存取 private field 跟 private method 會產生 compile time error 。

8.4 Referring to the Current Object's Members with the `this` Reference

- The `this` reference

別名

  - Any object can access a reference to itself with keyword `this`
  - Non-`static` methods implicitly use `this` when referring to the object's instance variables and other methods
  - Can be used to access instance variables when they are shadowed by local variables or method parameters

指區域變數或函數參數跟 field 命名相同的情形

- A `.java` file can contain more than one class
  - But only one class in each `.java` file can be `public`

Outline

ThisTest.java

(1 of 2)

```java
// Fig. 8.4: ThisTest.java
// this used implicitly and explicitly to refer to members of an object.

public class ThisTest
{
   public static void main( String args[] )
   {
      SimpleTime time = new SimpleTime( 15, 30, 19 );
      System.out.println( time.buildString() );
   } // end main
} // end class ThisTest

// class SimpleTime demonstrates the "this" reference
class SimpleTime
{
   private int hour;   // 0-23
   private int minute; // 0-59
   private int second; // 0-59

   // if the constructor uses parameter names identical to
   // instance variable names the "this" reference is
   // required to distinguish between names
   public SimpleTime( int hour, int minute, int second )
   {
      this.hour = hour;       // set "this" object's hour
      this.minute = minute;   // set "this" object's minute
      this.second = second;   // set "this" object's second
   } // end SimpleTime constructor

```

Create new **SimpleTime** object

Declare instance variables

Method parameters shadow instance variables

函數參數跟 Field 命名相同

Using this to access the object's instance variables

Outline

ThisTest.java

```
   // use explicit and implicit "this" to call toUniversalString
   public String buildString()
   {
      return String.format( "%24s: %s\n%24s: %s",
         "this.toUniversalString()", this.toUniversalString(),
         "toUniversalString()", toUniversalString() );
   } // end method buildString

   // convert to String in universal-time format (HH:MM:SS)
   public String toUniversalString()
   {
      // "this" is not required here to access instance variables,
      // because method does not have local variables with same
      // names as instance variables
      return String.format( "%02d:%02d:%02d",
         this.hour, this.minute, this.second );
   } // end method toUniversalString
} // end class SimpleTime
```

Using **this** explicitly and implicitly to call **toUniversalString**

(2 of 2)

同一個 Class 內部
不需要用到 this

Use of **this** not necessary here

```
this.toUniversalString(): 15:30:19
     toUniversalString(): 15:30:19
```

Error-Prevention Tip 8.1

Avoid method parameter names or local variable names that conflict with field names. This helps prevent subtle, hard-to-locate bugs.

養好習慣：

避免區域變數或函數參數跟 Field 命名相同的情形！！

8.5 Time Class Case Study: Overloaded Constructors

重覆定義

- **Overloaded constructors**
  - **Provide multiple constructor definitions with different signatures**

建構子

- **No-argument constructor**
  - **A constructor invoked without arguments**
- **The `this` reference can be used to invoke another constructor**
  - **Allowed only as the first statement in a constructor's body**

用 this 去呼叫其他建構子只允許放在該建構子的第一行

建構子的定義：跟 Class 同名，沒有回傳值的 Method 。

Outline

Time2.java

(1 of 4)

```java
// Fig. 8.5: Time2.java
// Time2 class declaration with overloaded constructors.

public class Time2
{
   private int hour;   // 0 - 23
   private int minute; // 0 - 59
   private int second; // 0 - 59

   // Time2 no-argument constructor: initializes each instance variable
   // to zero; ensures that Time2 objects start in a consistent state
   public Time2()
   {
      this( 0, 0, 0 ); // invoke Time2 constructor with three arguments
   } // end Time2 no-argument constructor

   // Time2 constructor: hour supplied, minute and second defaulted to 0
   public Time2( int h )
   {
      this( h, 0, 0 ); // invoke Time2 constructor with three arguments
   } // end Time2 one-argument constructor

   // Time2 constructor: hour and minute supplied, second defaulted to 0
   public Time2( int h, int m )
   {
      this( h, m, 0 ); // invoke Time2 constructor with three arguments
   } // end Time2 two-argument constructor

```

No-argument constructor

Invoke three-argument constructor

Outline

```
   // Time2 constructor: hour, minute and second supplied
   public Time2( int h, int m, int s )
   {
      setTime( h, m, s ); // invoke setTime to validate time
   } // end Time2 three-argument constructor

   // Time2 constructor: another Time2 object supplied
   public Time2( Time2 time )
   {
      // invoke Time2 three-argument c
      this( time.getHour(), time.getMinute(), time.getSecond() );
   } // end Time2 constructor with a Time2 object argument

   // Set Methods
   // set a new time value using universal time; ensure that
   // the data remains consistent by setting invalid values to zero
   public void setTime( int h, int m, int s )
   {
      setHour( h );   // set the hour
      setMinute( m ); // set the minute
      setSecond( s ); // set the second
   } // end method setTime

```

Call **`setTime`** method

Constructor takes a reference to another **`Time2`** object as a parameter

Could have directly accessed instance variables of object **`time`** here

`Time2.java`

(2 of 4)

```java
   // validate and set hour
   public void setHour( int h )
   {
      hour = ( ( h >= 0 && h < 24 ) ? h : 0 );
   } // end method setHour

   // validate and set minute
   public void setMinute( int m )
   {
      minute = ( ( m >= 0 && m < 60 ) ? m : 0 );
   } // end method setMinute

   // validate and set second
   public void setSecond( int s )
   {
      second = ( ( s >= 0 && s < 60 ) ? s : 0 );
   } // end method setSecond

   // Get Methods
   // get hour value
   public int getHour()
   {
      return hour;
   } // end method getHour

```

Time2

private int hour
private int minute
private int second

public Time2 (int)
public Time2 (int, int)
public Time2 (int, int, int)
public Time2 (Time2)

public void setTime (int, int, int)
public void setHour (int)
public void setMinute (int)
public void setSecond (int)
public int getHour ()
public int getMinute ()
public int getSecond ()
public String toUniversalString()
public String toString()

Outline

Time2.java

(4 of 4)

```
   // get minute value
   public int getMinute()
   {
      return minute;
   } // end method getMinute

   // get second value
   public int getSecond()
   {
      return second;
   } // end method getSecond

   // convert to String in universal-time format (HH:MM:SS)
   public String toUniversalString()
   {
      return String.format(
         "%02d:%02d:%02d", getHour(), getMinute(), getSecond() );
   } // end method toUniversalString

   // convert to String in standard-time format (H:MM:SS AM or PM)
   public String toString()
   {
      return String.format( "%d:%02d:%02d %s",
         ( (getHour() == 0 || getHour() == 12) ? 12 : getHour() % 12 ),
         getMinute(), getSecond(), ( getHour() < 12 ? "AM" : "PM" ) );
   } // end method toString
} // end class Time2
```

8.5 Time Class Case Study: Overloaded Constructors (Cont.)

- **Using *set* methods**
  - **Having constructors use *set* methods to modify instance variables instead of modifying them directly simplifies implementation changing**

建構子的重要工作：給定 Field 的初始值 (Initial Value) 。

Outline

```java
// Fig. 8.6: Time2Test.java
// Overloaded constructors used to initialize Time2 objects.

public class Time2Test
{
   public static void main( String args[] )
   {
      Time2 t1 = new Time2();          // 00:00:00
      Time2 t2 = new Time2( 2 );       // 02:00:00
      Time2 t3 = new Time2( 21, 34 );  // 21:34:00
      Time2 t4 = new Time2( 12, 25, 42 ); // 12:25:42
      Time2 t5 = new Time2( 27, 74, 99 ); // 00:00:00
      Time2 t6 = new Time2( t4 );      // 12:25:42

      System.out.println( "Constructed with:" );
      System.out.println( "t1: all arguments defaulted" );
      System.out.printf( "   %s\n", t1.toUniversalString() );
      System.out.printf( "   %s\n", t1.toString() );

```

Call overloaded constructors

Time2Test.java

```
      System.out.println(
         "t2: hour specified; minute and second defaulted" );
      System.out.printf( "   %s\n", t2.toUniversalString() );
      System.out.printf( "   %s\n", t2.toString() );

      System.out.println(
         "t3: hour and minute specified; second defaulted" );
      System.out.printf( "   %s\n", t3.toUniversalString() );
      System.out.printf( "   %s\n", t3.toString() );

      System.out.println( "t4: hour, minute and second specified" );
      System.out.printf( "   %s\n", t4.toUniversalString() );
      System.out.printf( "   %s\n", t4.toString() );

      System.out.println( "t5: all invalid values specified" );
      System.out.printf( "   %s\n", t5.toUniversalString() );
      System.out.printf( "   %s\n", t5.toString() );

```

```java
      System.out.println( "t6: Time2 object t4 specified" );
      System.out.printf( "   %s\n", t6.toUniversalString() );
      System.out.printf( "   %s\n", t6.toString() );
   } // end main
} // end class Time2Test
```

```
t1: all arguments defaulted
   00:00:00
   12:00:00 AM
t2: hour specified; minute and second defaulted
   02:00:00
   2:00:00 AM
t3: hour and minute specified; second defaulted
   21:34:00
   9:34:00 PM
t4: hour, minute and second specified
   12:25:42
   12:25:42 PM
t5: all invalid values specified
   00:00:00
   12:00:00 AM
t6: Time2 object t4 specified
   12:25:42
   12:25:42 PM
```

8.6 Default and No-Argument Constructors

- **Every class must have at least one constructor**
  - **If no constructors are declared, the compiler will create a default constructor**
    - **Takes no arguments and initializes instance variables to their initial values specified in their declaration or to their default values**
      - **Default values are zero for primitive numeric types, `false` for `boolean` values and `null` for references**
  - **If constructors are declared, the default initialization for objects of the class will be performed by a no-argument constructor (if one is declared)**

沒有參數的建構子又叫做_ Default Constructor 。
重複定義的建構子都會先呼叫它來做變數初始化動作。

8.10 Garbage Collection and Method `finalize`

- **Garbage collection** Java 特有的記憶體管理法
  - JVM marks an object for garbage collection when there are no more references to that object
  - JVM's garbage collector will retrieve those objects memory so it can be used for other objects
- **`finalize` method** 除構子 destructor
  - All classes in Java have the `finalize` method
    - Inherited from the `Object` class
  - `finalize` is called by the garbage collector when it performs termination housekeeping
  - `finalize` takes no parameters and has return type `void`

8.11 `static` Class Members

- **`static` fields**
  - Also known as class variables
  - Represents class-wide information
  - Used when:
    - all objects of the class should share the same copy of this instance variable or
    - this instance variable should be accessible even when no objects of the class exist
  - Can be accessed with the class name or an object name and a dot (.)
  - Must be initialized in their declarations, or else the compiler will initialize it with a default value (`0` for `int`s)

Outline

Employee.java

(1 of 2)

```java
// Fig. 8.12: Employee.java
// Static variable used to maintain a count of the number of
// Employee objects in memory.

public class Employee
{
   private String firstName;
   private String lastName;
   private static int count = 0; // number of objects in memory

   // initialize employee, add 1 to static count and
   // output String indicating that constructor was called
   public Employee( String first, String last )
   {
      firstName = first;
      lastName = last;

      count++;  // increment static count of employees
      System.out.printf( "Employee constructor: %s %s; count = %d\n",
         firstName, lastName, count );
   } // end Employee constructor

```

Declare a **static** field

宣告要給定初始值

Increment **static** field

Outline

Employee.java

```
   // subtract 1 from static count when garbage
   // collector calls finalize to clean up object;
   // confirm that finalize was called
   protected void finalize()
   {
      count--; // decrement static count of employees
      System.out.printf( "Employee finalizer: %s %s; count = %d\n",
         firstName, lastName, count );
   } // end method finalize

   // get first name
   public String getFirstName()
   {
      return firstName;
   } // end method getFirstName

   // get last name
   public String getLastName()
   {
      return lastName;
   } // end method getLastName

   // static method to get static count value
   public static int getCount()
   {
      return count;
   } // end method getCount
} // end class Employee
```

Declare method **`finalize`**

Declare **`static`** method **`getCount`** to get **`static`** field **`count`**

Employee
private String firstName private String lastName private static count
public Employee (String, String) protected void finalize ()
public String getFirstName() public String getLastName() public static int getCount()

Outline

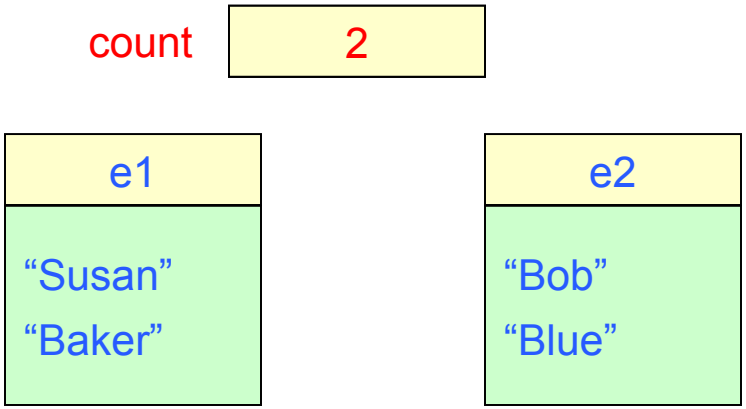
EmployeeTest.java

(1 of 3)

```java
// Fig. 8.13: EmployeeTest.java
// Static member demonstration.

public class EmployeeTest
{
   public static void main( String args[] )
   {
      // show that count is 0 before creating Employees
      System.out.printf( "Employees before instantiation: %d\n",
         Employee.getCount() );

      // create two Employees; count should be 2
      Employee e1 = new Employee( "Susan", "Baker" );
      Employee e2 = new Employee( "Bob", "Blue" );

```

Call **static** method **getCount** using class name **Employee**

Create new **Employee** objects



Outline

EmployeeTest.java

```
      // show that count is 2 after creating two Employees
      System.out.println( "\nEmployees after instantiation: " );
      System.out.printf( "via e1.getCount(): %d\n", e1.getCount() );
      System.out.printf( "via e2.getCount(): %d\n", e2.getCount() );
      System.out.printf( "via Employee.getCount(): %d\n",
         Employee.getCount() );

      // get names of Employees
      System.out.printf( "\nEmployee 1: %s %s\nEmployee 2: %s %s\n\n",
         e1.getFirstName(), e1.getLastName(),
         e2.getFirstName(), e2.getLastName() );

      // in this example, there is only one reference to each Employee,
      // so the following two statements cause the JVM to mark each
      // Employee object for garbage collection
      e1 = null;
      e2 = null;

      System.gc(); // ask for garbage collection to occur now
```

Call **static** method **getCount** inside objects

Call **static** method **getCount** outside objects

(2 of 3)

Remove references to objects, JVM will mark them for garbage collection

Call **static** method **gc** of class **System** to indicate that garbage collection should be attempted

Outline

EmployeeTest.java

(3 of 3)

```
      // show Employee count after calling garbage collector; count
      // displayed may be 0, 1 or 2 based on whether garbage collector
      // executes immediately and number of Employee objects collected
      System.out.printf( "\nEmployees after System.gc(): %d\n",
         Employee.getCount() );
   } // end main
} // end class EmployeeTest
```

Call **static** method **getCount**

```
Employees before instantiation: 0
Employee constructor: Susan Baker; count = 1
Employee constructor: Bob Blue; count = 2

Employees after instantiation:
via e1.getCount(): 2
via e2.getCount(): 2
via Employee.getCount(): 2

Employee 1: Susan Baker
Employee 2: Bob Blue

Employee finalizer: Bob Blue; count = 1
Employee finalizer: Susan Baker; count = 0

Employees after System.gc(): 0
```

定義在建構子中

定義在除構子中

8.11 `static` Class Members (Cont.)

- `String` objects are immutable
  - `String` concatenation operations actually result in the creation of a new `String` object
- `static` method `gc` of class `System`
  - Indicates that the garbage collector should make a best-effort attempt to reclaim objects eligible for garbage collection
  - It is possible that no objects or only a subset of eligible objects will be collected
- `static` methods cannot access non-`static` class members
  - Also cannot use the `this` reference

8.13 `final` Instance Variables

- **Principle of least privilege**
  - **Code should have only the privilege and access it needs to accomplish its task, but no more**
- **`final` instance variables**
  - **Keyword `final`**
    - **Specifies that a variable is not modifiable (is a constant)**
  - **`final` instance variables can be initialized at their declaration**
    - **If they are not initialized in their declarations, they must be initialized in all constructors**

Outline

Increment.java

total 可變，INCREMENT 不可變

```java
// Fig. 8.15: Increment.java
// final instance variable in a class.

public class Increment
{
   private int total = 0; // total of all increments
   private final int INCREMENT; // constant variable (uninitialized)

   // constructor initializes final instance variable INCREMENT
   public Increment( int incrementValue )
   {
      INCREMENT = incrementValue; // initialize constant variable (once)
   } // end Increment constructor

   // add INCREMENT to total
   public void addIncrementToTotal()
   {
      total += INCREMENT;
   } // end method addIncrementToTotal

   // return String representation of an Increment object's data
   public String toString()
   {
      return String.format( "total = %d", total );
   } // end method toIncrementString
} // end class Increment
```

Declare **final** instance variable

Initialize **final** instance variable inside a constructor

Increment
private int total private final int INCREMENT
public Increment (int)
public void addIncrementToTotal() public String toString()

Outline

IncrementTest.java

```java
// Fig. 8.16: IncrementTest.java
// final variable initialized with a constructor argument.

public class IncrementTest
{
   public static void main( String args[] )
   {
      Increment value = new Increment( 5 );

      System.out.printf( "Before incrementing: %s\n\n", value );

      for ( int i = 1; i <= 3; i++ )
      {
         value.addIncrementToTotal();
         System.out.printf( "After increment %d: %s\n", i, value );
      } // end for
   } // end main
} // end class IncrementTest
```

Create an **Increment** object

Call method **addIncrementToTotal**

```
Before incrementing: total = 0

After increment 1: total = 5
After increment 2: total = 10
After increment 3: total = 15
```

Increment
private int total private final int INCREMENT
public Increment (int)
public void addIncrementToTotal() public String toString()

8.16 Time Class Case Study: Creating Packages

- To declare a reusable class
  - Declare a `public` class
  - Add a `package` declaration to the source-code file
    - must be the very first executable statement in the file
    - `package` name should consist of your Internet domain name in reverse order followed by other names for the package
      - example: `com.deitel.jhtp6.ch08`
      - `package` name is part of the fully qualified class name
        - Distinguishes between multiple classes with the same name belonging to different packages
        - Prevents name conflict (also called name collision)
      - Class name without `package` name is the simple name

Outline

```java
// Fig. 8.18: Time1.java
// Time1 class declaration maintains the time in 24-hour format.
package com.deitel.jhtp6.ch08;

public class Time1
{
   private int hour;   // 0 - 23
   private int minute; // 0 - 59
   private int second; // 0 - 59

   // set a new time value using universal time; perform
   // validity checks on the data; set invalid values to zero
   public void setTime( int h, int m, int s )
   {
      hour = ( ( h >= 0 && h < 24 ) ? h : 0 );   // validate hour
      minute = ( ( m >= 0 && m < 60 ) ? m : 0 ); // validate minute
      second = ( ( s >= 0 && s < 60 ) ? s : 0 ); // validate second
   } // end method setTime

```

package declaration

Time1 is a **public** class so it can be used by importers of this package

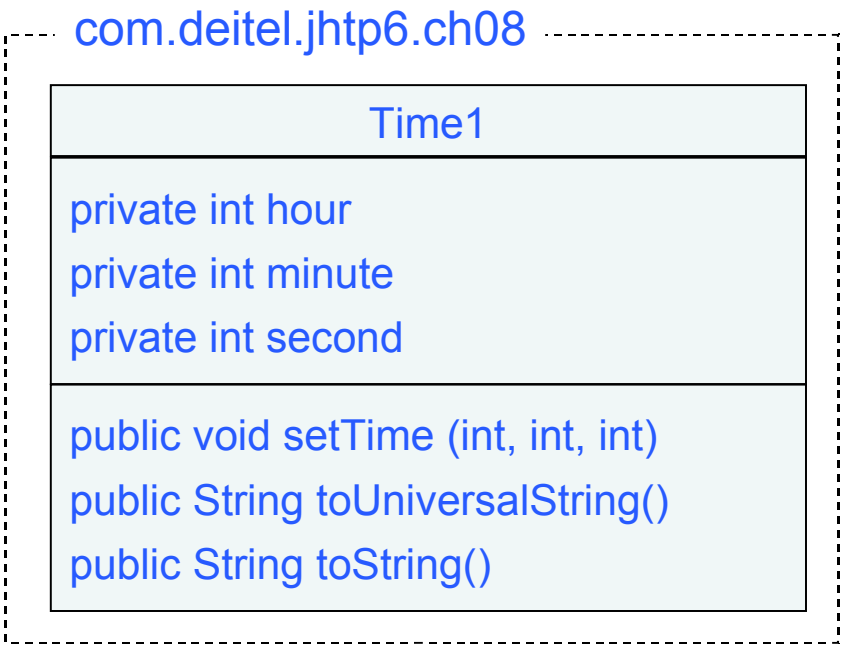
Time1.java

Outline

Time1.java

(2 of 2)

```java
   // convert to String in universal-time format (HH:MM:SS)
   public String toUniversalString()
   {
      return String.format( "%02d:%02d:%02d", hour, minute, second );
   } // end method toUniversalString

   // convert to String in standard-time format (H:MM:SS AM or PM)
   public String toString()
   {
      return String.format( "%d:%02d:%02d %s",
         ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 ),
         minute, second, ( hour < 12 ? "AM" : "PM" ) );
   } // end method toString
} // end class Time1
```



8.16 Time Class Case Study: Creating Packages (Cont.)

- Compile the class so that it is placed in the appropriate package directory structure
  - Example: our package should be in the directory

```
com
  └──→ deitel
         └──→ jhtp6
               └──→ ch08
```

  - `javac` command-line option `-d`
    - `javac` creates appropriate directories based on the class's `package` declaration
    - A period (.) after `-d` represents the current directory

練習： javac –d . Time1.java

8.16 Time Class Case Study: Creating Packages (Cont.)

- Import the reusable class into a program
  - Single-type-import declaration
    - Imports a single class
    - Example: `import java.util.Random;`
  - Type-import-on-demand declaration
    - Imports all classes in a package
    - Example: `import java.util.*;`

class name

package name

Outline

```
// Fig. 8.19: Time1PackageTest.java
// Time1 object used in an application.
import com.deitel.jhtp6.ch08.Time1; // import class Time1

public class Time1PackageTest
{
   public static void main( String args[] )
   {
      // create and initialize a Time1 object
      Time1 time = new Time1(); // calls Time1 constructor

      // output string representations of the time
      System.out.print( "The initial universal time is: " );
      System.out.println( time.toUniversalString() );
      System.out.print( "The initial standard time is: " );
      System.out.println( time.toString() );
      System.out.println(); // output a blank line

```

class name

Single-type **import** declaration

package name

Refer to the **Time1** class by its simple name

Time1PackageTest.java

(1 of 2)

Outline

```
      // change time and output updated time
      time.setTime( 13, 27, 6 );
      System.out.print( "Universal time after setTime is: " );
      System.out.println( time.toUniversalString() );
      System.out.print( "Standard time after setTime is: " );
      System.out.println( time.toString() );
      System.out.println(); // output a blank line

      // set time with invalid values; output updated time
      time.setTime( 99, 99, 99 );
      System.out.println( "After attempting invalid settings:" );
      System.out.print( "Universal time: " );
      System.out.println( time.toUniversalString() );
      System.out.print( "Standard time: " );
      System.out.println( time.toString() );
   } // end main
} // end class Time1PackageTest
```

```
The initial universal time is: 00:00:00
The initial standard time is: 12:00:00 AM

Universal time after setTime is: 13:27:06
Standard time after setTime is: 1:27:06 PM

After attempting invalid settings:
Universal time: 00:00:00
Standard time: 12:00:00 AM
```

Time1PackageTest.java

(2 of 2)

〔練習一〕

以 8.5, 8.6 為例，分別為程式加上 finalize() 及 System.gc().

Time2

private int hour
private int minute
private int second

public Time2 (int)
public Time2 (int, int)
public Time2 (int, int, int)
public Time2 (Time2)
protected void finalize()

public void setTime (int, int, int)
public void setHour (int)
public void setMinute (int)
public void setSecond (int)
public int getHour ()
public int getMinute ()
public int getSecond ()
public String toUniversalString()
public String toString()

〔練習二〕

承上，為程式加上 static field count 及 static method getCount()

Time2
private int hour private int minute private int second private static int count
public Time2 (int) public Time2 (int, int) public Time2 (int, int, int) public Time2 (Time2) protected void finalize()
public void setTime (int, int, int) public void setHour (int) public void setMinute (int) public void setSecond (int) public int getHour () public int getMinute () public int getSecond () public String toUniversalString() public String toString() public static int getCount()

〔練習三〕

(1) 承上，加上 package 宣告， package name 為 org.test.sample ，並嘗試使用 javac –d . Time2.java 來產生 class 檔。

(2) 嘗試使用以下指令執行所產生的 class 檔。
java –classpath …

Time2
private int hour private int minute private int second private static int count
public Time2 (int) public Time2 (int, int) public Time2 (int, int, int) public Time2 (Time2) protected void finalize()
public void setTime (int, int, int) public void setHour (int) public void setMinute (int) public void setSecond (int) public int getHour () public int getMinute () public int getSecond () public String toUniversalString() public String toString() public static int getCount()