Data Structures for Java

William H. Ford
William R. Topp

Classes and Objects

Bret Ford

Data Structures

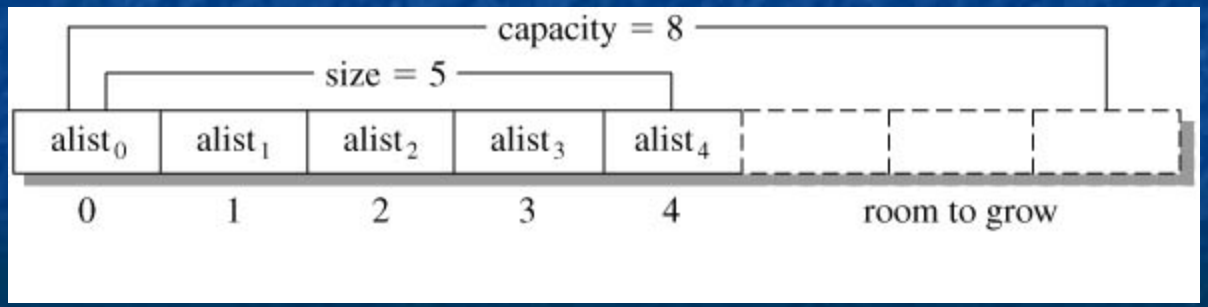
- Storage processing machines that handle large sets of data. The structures, called collections, have operations that add and remove items and provide access to the elements in the collection.

Arrays

- Data structures that store a large collection of data and allow direct access to the elements by using an index.
  - Have a fixed length. A programmer must specify the size of an array at the point of its declaration.
  - Applications often need storage structures, which grow and contract as data is added and removed.

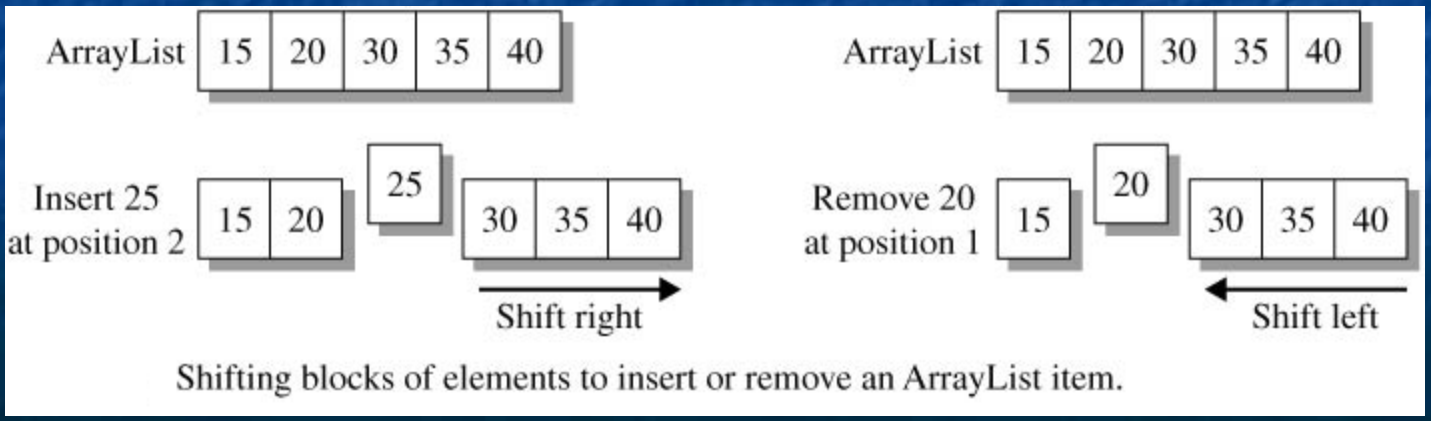
ArrayList

- Indexing features of an array
- Created with initial capacity that designates the available space to hold elements.
- Ability to grow dynamically to meet the runtime needs of a program



ArrayList (concluded)

- Not an efficient storage structure for general insertion and deletion of items at an arbitrary position in a list.
  - Insert requires shifting a block of elements to the right to make room for a new item.
  - Deletion requires shifting a block of elements to the left to fill in the gap created by the missing item.



Shifting blocks of elements to insert or remove an ArrayList item.

Data Structures and Object-Oriented Programming

- Modern object-oriented programming is an ideal mechanism for designing and implementing a data structure.
- A collection is an object and the collection type (class) is the data structure.
  - The object has private members that define the data.
  - The public methods provide the data handling operations.

Data Structures and Object-Oriented Programming (continued)

- Java has good object-oriented design constructs that support a modern view of data structures.

Data Structures and Algorithms

- Data structures have operations that manipulate the data.
- The design and implementation of these operations involve algorithms that consist of a finite sequence of instructions to perform a task.

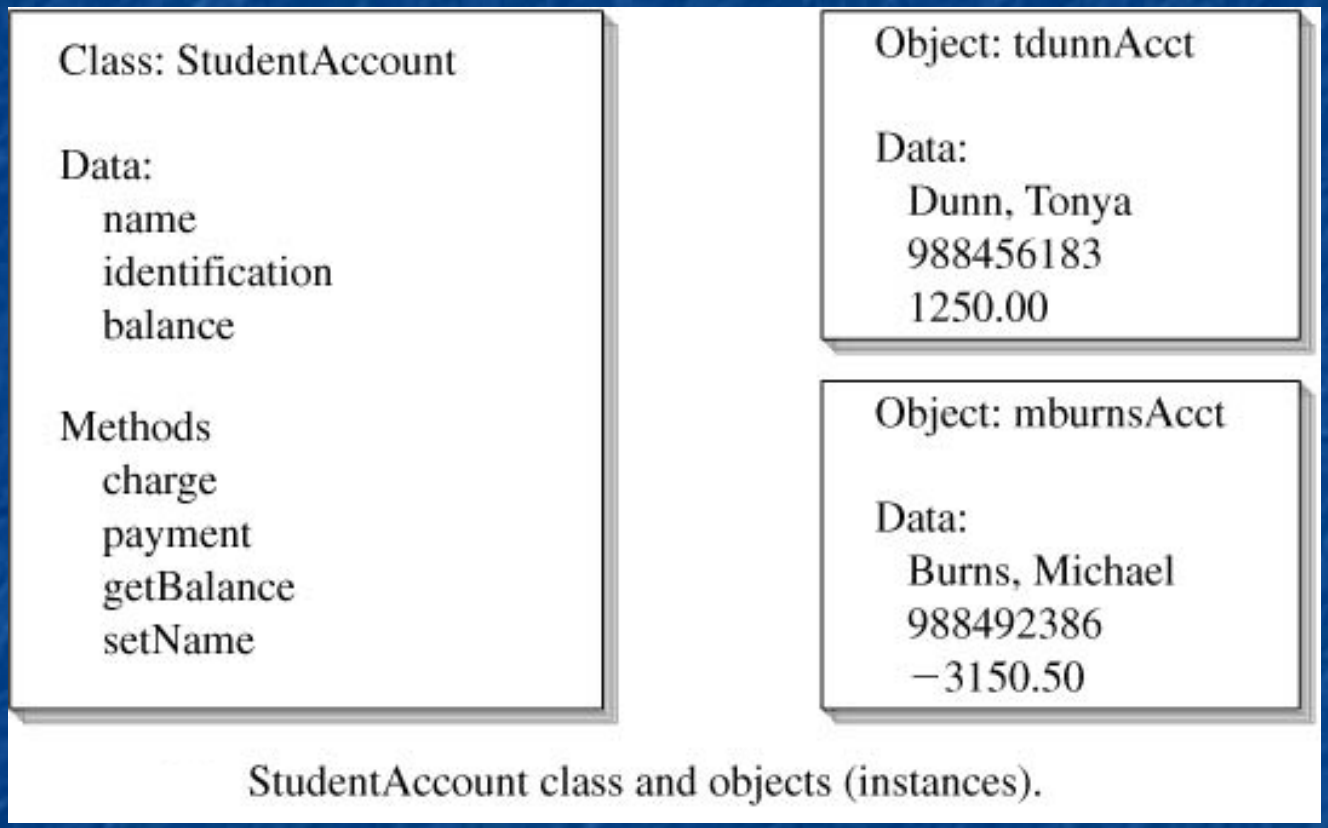
Data Structures and Algorithms (concluded)

- The interplay between algorithms and data structures was highlighted by the renowned teacher and scholar, Nicolas Wirth, who coined the expression "data structures + algorithms = programming."

Object-Oriented Programming

- An object is an entity with data and operations on the data.
- A class is the type of an object and describes what the object can do.
- Objects are instances of the class.
- Creating a computer program using objects as the starting point is called *object-oriented programming*.

StudentAccount Class



StudentAccount class and objects (instances).

Understanding a Class

- Designing a class involves specifying data and methods that act on the data that allow a programmer to create and use an object in a problem situation.
- The implementation makes the operations of a class work.

Class Methods

- A method accepts data values called *arguments*, carries out calculations, and returns a value.
- A method may have *preconditions* that must apply in order that the operation properly executes.
- If a precondition is violated, the method identifies the condition and throws an exception, which allows the calling statement to employ some error handling strategy.

Class Methods (concluded)

- A method also has *postconditions* that indicate changes to the object's state caused by executing the method.

Method Signature

- Includes the method name, a listing of formal parameters that correspond to arguments, and the object type of the return value.
  - The parameter list is a comma-separated sequence of one or more parameter names and data types.

Method Signature (concluded)

- May also include an action statement that describes the operation and any preconditions and any postconditions exist.

```
ReturnType methodName(Type1 param1, Type2 param2, ...)
    Action statement
```

Time24 Class Design

- A Time24 object has integer data variables, hour and minute, where the hour is in the range 0 to 23 and minute is in the range 0 to 59.
- The interface has methods to access and update the data members and to create a formatted string that describes the current time.

Time24 Methods

addTime:

```
void addTime(int m)
```

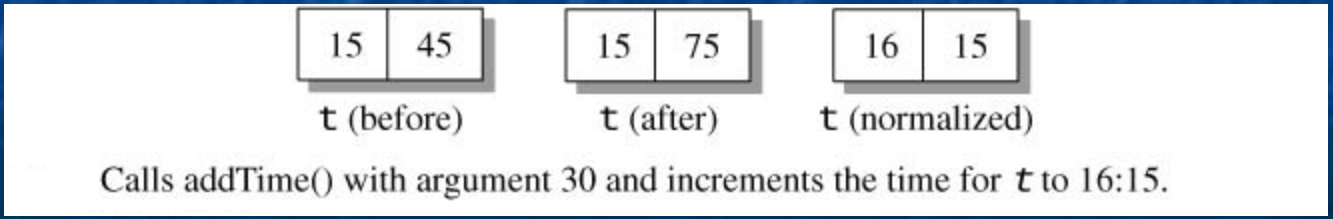
Updates the time for this object by the specified number of minutes. The hour and minute values are normalized so that hour is in the range 0 to 23 and minute is in the range 0 to 59.

Example: **Assume t is a Time24 object with time 15:45 (3:45 PM).**

The statement

```
t.addTime(30);
```

calls addTime with argument 30 and increments the time for t to 16:15.



Calls addTime() with argument 30 and increments the time for t to 16:15.

Time24 Methods (continued)

The method interval(t) measures the time gap between the current hour and minute of the object and next 24-hour occurrence of a Time24 argument t. The result is a Time24 object that becomes the return value.

interval:

```
Time24 interval(Time24 t)
  Returns a Time24 object that specifies length of time
  from the current hour and minute to the next 24-hour
  occurrence of time t.
```

Time24 Methods (concluded)

***Example*: Use Time24 objects tA, tB, and tC. Object tA has current time 14:45 (2:45 PM) and object tB has current time 16:15 (4:15 PM).**

Case 1: `tC = tA.interval(tB)` `tC` is the interval from `tA` to `tB`

14	45

`tA`

16	15

`tB`

1	30

`tC`

Case 2: `tC = tB.interval(tA)` `tC` is the interval from `tB` to `tA`

16	15

`tB`

14	45

`tA`

22	30

`tC`

Constructors

- A *constructor* allocates memory for an object and initializes the data values.
- The method signature for a constructor uses the class name as the method name and does not have a return type.
- A class can provide a variety of constructors by using different parameter lists.
- The *default constructor* that has an empty parameter list and assigns default values to the data.

Time24 Constructors

`Time24()`

The default constructor initializes the hour and minute to be 0. The time is midnight (0:00)

`Time24(int hour, int minute)`

The object is initialized with the specified hour and minute. The values of the data members are normalized to their proper range.

Example:

```
Time24()         // object is 0:00
Time24(4,45)     // object is 4:45

// 150 minutes is 2:30
// hour 23 and minute 150 are normalized
Time24(23,150) // object value is 1:30
```

The toString Method

- A class typically provides a method, toString(), which describes the state of the object.
  - The return type is a String containing some formatted representation of the object.

Time24 class

toString():

```
String toString()
```

Returns a string that describes the time of the object in the form hh:mm

Time24 Class toString Example

Assume t is a Time24 object with hour 14 and minute 45. Use the method toString() and System.out.print() for console output.

```
System.out.print("Time t is " + t.toString());
```

```
Output: Time t is 14:45
```

Getter Methods

- In general, a program may not directly access the data in an object.
- An *getter method* returns the current value for an object's data field and typically begins with the word *get*.

Time24 Class Getters

getHour():

```
int getHour()
```

Returns the hour value for the object

getMinute():

```
int getMinute()
```

Returns the minute value for the object

Setter Methods

- A *setter method* allows a program to update the value for one or more data fields in the object. The method name typically begins with the word *set*

Time24 Class Setter

setTime:

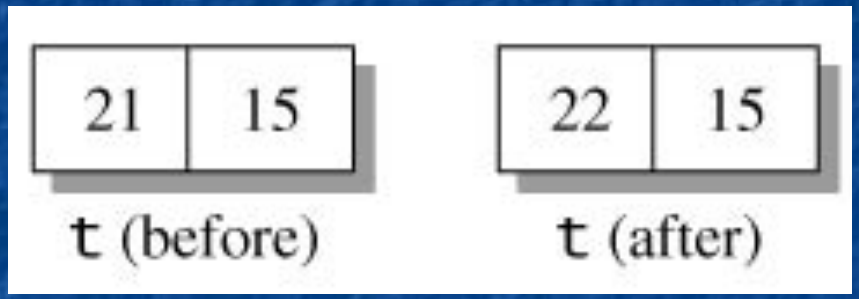
```
void setTime(int hour, int minute)
```

The arguments update the corresponding data values in the object.

Setter Methods (concluded)

Example: The methods getHour(), getMinute(), and setTime() combine to move the clock forward one hour This is equivalent to addTime(60).

```
t.setTime(t.getHour() + 1, t.getMinute())
```



Static Methods

- Object methods access and update the data fields of an object.
- The design of a class can include independent methods that are not bound to any specific object. Their signatures include the modifier *static*.
- These operations are referred to as *class* or *static methods.* They are called by using the class name to reference the method.

Time24 Static Method

Time24 Example:

parseTime():

```
static Time24 parseTime(String s)
```

The method takes a string s in the form hh:mm and returns an object with the equivalent Time24 value.

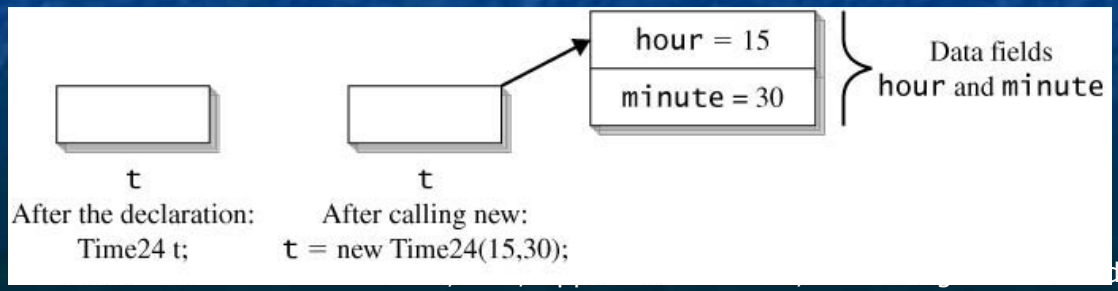
```
// t has value 13:45 (1:45 PM)
t = Time24.parseTime("13:45");
```

Declaring Objects

- You must make a reference variable refer to an actual object y assigning it a value returned by the new operator that calls a constructor to create an object.

Example:

```
Time24 t;
// creates object with time midnight
t = new Time24();
// creates object with time 15:30
t = new Time24(15,30);
```



Using Object Methods

- Once an object is created, a program can call any of its interface methods.
- The syntax uses the object reference and the method name separated by a "." (dot). The method should include zero or more arguments enclosed in parentheses.

```
// call addTime(45) which advances time for t
// 45 minutes; t becomes 16:15
t.addTime(45)
```

The Keyword null

- An object reference variable can be initially set to null.
- The value is a special constant that can be used to give a value to any variable of class type. It indicates that the variable does not yet reference an actual object.
- The runtime system identifies any attempt to call an object method when the reference variable has value null as an error.

References and Aliases

- An assignment statement with primitive variables copies the value of the right-side to the left-side variable. The result is two variables in separate memory locations but with the same value.

```
int m = 5, n;
n = m;
```

References and Aliases (concluded)

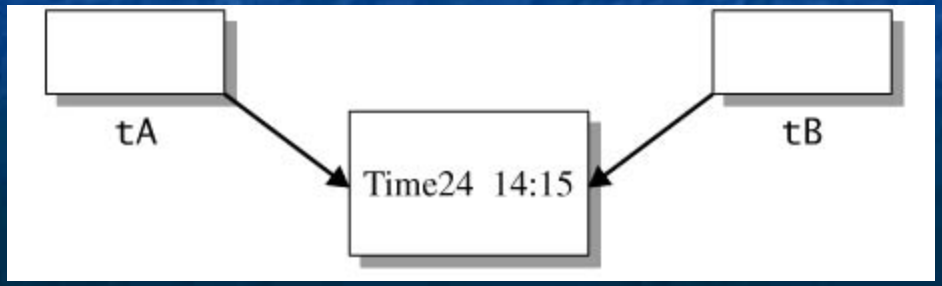
- Assignment also applies to reference variables, but the results are different from assignment of primitive variables.

Assignment of Reference Variables

```
Time24 tA = new Time24(14,15), tB;

tB = tA;
```

The result is two separate reference variables tA and tB both pointing to the same object. Using any update method with either variable changes the state of the single object they reference.



PROGRAM 1-1
Time24 Broadcast Times

```
import ds.time.Time24;

public class Program1_1
{
   public static void main(String[] args)
   {
      // Time24 reference variables
      // game 1:15 PM
      Time24 startGame = new Time24(13,15),
      // length 3:23
      timeOfGame = new Time24(3,23),
      // news 5:00 PM
      startNews = Time24.parseTime("17:00");
      // uninitialized
      Time24 endGame, timeForInterviews;
```

PROGRAM 1-1 (continued)

```
// create object for endGame, with same time
 // as startGame; update it by adding the
 // time for the game in minutes.
endGame =
    new Time24(startGame.getHour(),
               startGame.getMinute());

// declare an integer that stores length of
// game in minutes
int minutesOfGame =
     timeOfGame.getHour()*60 +
     timeOfGame.getMinute();
```

PROGRAM 1-1 (continued)

```
// advance time of endGame using addTime()
 endGame.addTime(minutesOfGame);

 // assign interval() to variable
 // timeForInterviews
 timeForInterviews =
    endGame.interval(startNews);
```

PROGRAM 1-1 (concluded)

```
        // output
        System.out.println("The game begins at " +
                           startGame);
        System.out.println("The game ends at " +
                           endGame);
        System.out.println("Post game interviews" +
                           last " +
                           timeForInterviews);
    }
}
```

```
Run:

The game begins at 13:15
The game ends at 16:38
Post game interviews last 0:22
```

API (Application Programming Interface)

- Summary of the class design.
- Begins with a header that includes the class name and the package name.
- Separate listing of class constructors and interface methods.
- Allows an application programmer to create a class object and employ its methods in a program.

API (concluded)

- Programmer can use an API to understand a class as a toolkit without reference to its implementation details.

Time24 Class API

class Time24	ds.time
	Constructors
	Time24() The constructor initializes the hour and minute to be 0. The time is midnight (0:00)
	Time24(int hour, int minute) The object is initialized with specified hour and minute. The values of the data fields are normalized to their proper range. If hour or minute are negative, it throws an IllegalArgumentException

Time24 Class API (continued)

	Methods
void	**addTime**(int m) Advances the time for this object by the specified number of minutes. The resulting hour and minute are normalized to their proper range. If m is negative, the method throws an IllegalArgumentException.
int	**getHour**() Returns the hour value
int	**getMinute**() Returns the minute value
Time24	**interval**(Time24 t) Returns a Time24 object that specifies length of time from the current hour and minute to the next 24-hour occurrence of time t.

Time24 Class API (concluded)

static Time24	**parseTime**(String str) The method takes a string s in the form hh:mm and returns an object with the equivalent Time24 value.
void	**setTime**(int hour, int minute) The arguments update the data fields of the object and normalize them to their proper range. Both arguments must be positive integer values; otherwise, the method throws an IllegalArgumentException.
String	**toString**() Returns a string that describes the state of the object in the form hh:mm

Unified Modeling Language (UML)

- Gives a graphical representation of classes and their interaction.
- Has three compartments that provide the class name, the instance variables and their types, and the signature for each method.
- Each member of the class begins with the symbol '+' or '-' indicating the visibility modifiers public and private, respectively

UML Representation of the Time24 Class

Time24
−hour: int −minute: int
+Time24() +Time24(hour: int, minute: int) +addTime(m: int): void +getHour(): int +getMinute(): int +interval(t:Time24): Time24 −normalizeTime(): void +parseTime(s:String): String +setTime(hour: int, minute: int): void +toString()(): String

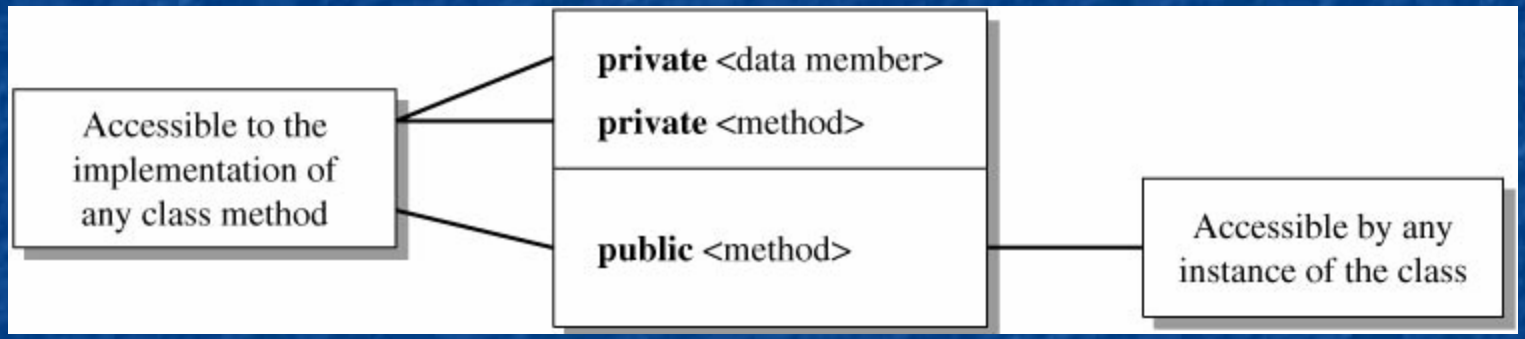
Information Hiding

- Visibility modifiers *private* and *public*.
- A visibility modifier is included in the declaration of the class header and in the declaration of each class member.
- The public modifier in the class header indicates that the class, and hence instances, are available to an application class outside of the package.

Information Hiding (concluded)

- For class members, the public modifier specifies that the member is accessible by any instance of the class. The public members represent the class interface.
- The private modifier typically applies to the data in the class and to utility methods that support the class implementation. Only methods of the class can access private members.

Access Rights to Public and Private Members of a Class



Time24 Class Declaration

```
public class Time24
{
   // hour in the range 0 to 23
   private int hour;
   // minute in the range 0 to 59
   private int minute;
   // utility method sets hour and minute in
   // their proper range
   private void normalizeTime() { . . . }
   // constructors
   public Time24() { . . . }
   public Time24(int hour, int minute) { . . .  }
   // methods that increment time and identify
   // time intervals
   public void addTime(int m) { . . . }
   public Time24 interval(Time24 t) { . . . }
```

Time24 Class Declaration (concluded)

```
    // getter and setter methods for the two
    // variables
    public int getHour() { . . . }
    public int getMinute() { . . . }
    public void setTime(int hour, int minute)
    { . . . }
    // string that describes the object
    public String toString() { . . . }
    // static method converts a string to a
    // Time24 object
    public static Time24 parseTime(String s)
    { . . . }
}
```

Private Methods

- Make the implementation of public methods more efficient and easier to read.
- Used only within the class implementation and are not part of the interface.
- Decompose a complex public method into subtasks which are handled by private methods.
- Use a private method to carry out a task that must be repeated many times during the implementation of the class.

Time24 Class Private Method

normalizeTime():

```
// utility method sets the hour value
// in the range 0 to 23 and the minute
// value in the range 0 to 59
private void normalizeTime()
{
   int extraHours = minute / 60;
   // set minute in range 0 to 59
   minute %= 60;
   // update hour. set in range 0 to 23
   hour = (hour + extraHours) % 24;
}
```

Time24 Class Getter Methods

***getHour()*:**

```
// return the current value of the
// instance variable hour
public int getHour()
{ return hour; }
```

***getMinute()*:**

```
// return the current value of the
// instance variable minute
public int getMinute()
{ return minute; }
```

Time24 Class Setter Method

```
public void setTime(int hour,
                    int minute)
{
   // check that arguments hour and
   // minute are positive
   if (hour < 0 || minute < 0)
      throw  new
        IllegalArgumentException
          ("Time24 setTime: " +
           "parameters must be" +
           "positive integers");
```

Time24 Class Setter Method (concluded)

```
        // assign new values
        // normalizeTime()
        this.hour = hour;
        this.minute = minute;
        // adjust hour and minute as necessary
        normalizeTime();
    }
```

Constructor

- A public method that uses the class name to identify the method and may not have a return type.
- Has a parameter list specifying arguments that initialize the object.

Time24 Constructors

***Constructor Time24(h,m)*:**

```
// constructor creates an instance
// with values hour and minute.
public Time24(int hour, int minute)
{
   setTime(hour,minute);
}
```

***Default constructor Time24()*:**

```
// constructor creates an instance
// set to midnight
public Time24()
{
   this(0,0);
}
```

Time24 toString()

Time24 toString():

```
public String toString()
{
    // create a text format object
    // with two character positions
    // and fill character 0
    DecimalFormat fmt =
        new DecimalFormat("00");
    return new String(hour + ":" +
        fmt.format(minute));
}
```

Incrementing Time

addTime():

```
public void addTime(int m)
{
    if (m < 0)
        throw new
     IllegalArgumentException
     ("Time24.setTime: argument"
+
       " must be a positive " +
       "integer");
    minute += m;
    normalizeTime();
}
```

Time24 Interval

```
interval():
// return the length of time from the current time to
// time t
public Time24 interval(Time24 t)
{
   // convert current time and time
   // t to minutes
   int currTime = hour * 60 +
                  minute;
   int tTime = t.hour * 60 +
               t.minute;
```

Time24 Interval (concluded)

```
    // if t is an earlier time, then
    // add 24 hours so that t
    // represents a time in the
    // "next day"
    if (tTime < currTime)
        tTime += 24 * 60;
    // return a reference to a new
    // Time24 object
    return
        new Time24(0, tTime-currTime);
}
```

Package

- Group of related classes stored in a directory and made accessible to programs using *import* statements.
- Promotes code reuse.

Package (concluded)

- Place a class in a package by adding a package header at the beginning of the source code file and then placing the file in a directory with the same name as the package name.

```
package DemoPackage;

public class DemoClass
{ . . . }
```

Importing a Class

- Import a specific class

```
import DemoPackage.DemoClass;
```

- Import all classes in a package

```
import DemoPackage.*;
```

The Java Software Development Kit

- java.lang - Contains the basic classes needed for developing Java programs. This includes the String class and the System class.
- java.util - Provides the Collections Framework classes, the Random class, the StringTokenizer class, and classes dealing with dates and times.

The Java Software Development Kit (concluded)

- java.io - Classes manage the input and output of data.
- Many other packages.

Strings

- A sequence of characters that programs use to identify names, words, and sentences.
- String class in the "java.lang" package.

String Literals

```
String city = "Phoenix";
String[] weekdays = {"Mon", "Tue", "Wed", "Thu", "Fri"};
```

String Indexing

```
String str = "this string has vowels";
char ch;
System.out.println("Length is " +
                   str.length());
for (int i=0; i < str.length(); i++)
{
   ch = str.charAt(i);
   if (ch == 'a' || ch == 'e' ||
       ch == 'i' || ch == 'o' || ch == 'u')
      System.print(ch + " ");
}
```

```
Output: Length is 22
        i i a o e
```

String Concatentation

```
String strA = "down", strB = "town", strC;

strC = strA + strB;  // strC is "downtown"
// strA is "down periscope"
strA += " periscope";
```

String Comparison using equals()

```
String strA = "goodbye";
boolean b;

// b is true
b = strA.equals("good"+"bye");

// b is false
b = strA.equals("goodby");
```

String Comparison using compareTo()

Return value from compareTo(anotherStr)

$$\begin{cases} <0 & thisStr < anotherStr \\ =0 & thisStr == anotherStr \\ >0 & thisStr > anotherStr \end{cases}$$

```
String s1 = "John", s2 = "Johnson";

// John < Johnson is true
s1.compareTo(s2) < 0
// Johnson >= Mike is false
s2.compareTo("Mike") >= 0
// John == John is true
s1.compareTo("John") == 0
```

Enum Type

- Facility that allows a program to define a special kind of class whose members are named constants.
- The class has a simple declaration the uses the keyword enum followed by the name and a list of identifiers enclosed in braces.

```
public enum Season {fall, winter, spring, summer};
```

Enum Types (concluded)

- An enum type has the method toString() which returns the constant name and the operator == which determines whether two values are the same.
- The method compareTo() compares two enum values using the order of names in the enum declaration.

Enum Example

```
// declare two variables of type enum Season
// and initialize the variables
Season s = Season.winter, t = Season.spring;

// equals() tests the value of s
if (s == Season.winter || s == Season.spring)
   System.out.println("In " + s +
                      " economy fares apply");

// compareTo() affirms the ordering of the
// seasons
if (s.compareTo(t) < 0)
   System.out.println(s + " comes before " + t);
```

```
Output:
   In winter economy fares apply
   winter comes before spring
```