Abstract Classes

Introduction to Programming and Computational Problem Solving: Accelerated Pace
CSE 11
Lecture 15
Announcements

• Assignment 6 is due today, 11:59 PM
  – Upgrade beginning May 25, 12:01 AM
• Assignment 7 will be released today
  – Due May 30, 11:59 PM
Abstract classes

• Remember, a superclass defines common behavior for related subclasses
  – Inheritance enables you to define a general class (i.e., a superclass) and later extend it to more specialized classes (i.e., subclasses)
• Sometimes, a superclass is so general it cannot be used to create objects
  – Such a class is called an abstract class
• An abstract class can contain abstract methods that are implemented in concrete subclasses
• Just like nonabstract classes, models is-a relationships
  – For example
    • Circle is-a GeometricObject
    • Rectangle is-a GeometricObject
Abstract class example

**GeometricObject**

- color: String
- filled: boolean
- dateCreated: java.util.Date

#GeometricObject()
#GeometricObject(color: string, filled: boolean)
+ getColor(): String
+ setColor(color: String): void
+ isFilled(): boolean
+ setFilled(filled: boolean): void
+ getDateCreated(): java.util.Date
+ toString(): String
  + getArea(): double
  + getPerimeter(): double

The # sign indicates protected modifier

Methods getArea and getPerimeter are overridden in Circle and Rectangle. Superclass methods are generally omitted in the UML diagram for subclasses.

**Circle**

- radius: double
  + Circle()
  + Circle(radius: double)
  + Circle(radius: double, color: string, filled: boolean)
  + getRadius(): double
  + setRadius(radius: double): void
  + getDiameter(): double

**Rectangle**

- width: double
- height: double
  + Rectangle()
  + Rectangle(width: double, height: double)
  + Rectangle(width: double, height: double, color: string, filled: boolean)
  + getWidth(): double
  + setWidth(width: double): void
  + getHeight(): double
  + setHeight(height: double): void
Unified Modeling Language (UML)

+ public
# protected
- private

- Static variables and methods are underlined
- Abstract class names and methods are italicized
- Interface names and methods are italicized
- Open or no arrow is association
- Hollow diamond is aggregation
- Filled diamond is composition
- Hollow triangle is inheritance
- Dashed line with hollow triangle is implementation of interface

Covered later in the quarter
## Methods and data fields visibility

<table>
<thead>
<tr>
<th>Modifiers on Members in a Class</th>
<th>Accessed from the Same Class</th>
<th>Accessed from the Same Package</th>
<th>Accessed from a Subclass in a Different Package</th>
<th>Accessed from a Different Package</th>
</tr>
</thead>
<tbody>
<tr>
<td>Public</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Protected</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Default (no modifier)</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Private</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Abstract classes and abstract methods are denoted using the `abstract` modifier.

Example:
```java
public abstract class GeometricObject {
    private String color = "white";
    private boolean filled;
    private java.util.Date dateCreated;

    protected GeometricObject() {
        dateCreated = new java.util.Date();
    }

    protected GeometricObject(String color, boolean filled) {
        dateCreated = new java.util.Date();
        this.color = color;
        this.filled = filled;
    }

    // Abstract method
    public abstract double getArea();

    // Abstract method
    public abstract double getPerimeter();
}
```

 Constructors in an abstract class are protected because they are only used by subclasses.
Abstract methods are only allowed in abstract classes

• An abstract method cannot be contained in a nonabstract class

• If a subclass of an abstract superclass does not implement all the abstract methods, then the subclass must be defined abstract

• In other words, in a **nonabstract subclass** extended from an abstract class, **all** the abstract methods must be implemented, **even if they are not used in the subclass**
An object cannot be created from an abstract class

• An abstract class cannot be instantiated using the new operator

• You can still define its constructors, which are invoked in the constructors of its subclasses
  – For example, the constructors of GeometricObject are invoked in the Circle class and the Rectangle class
An abstract class without any abstract methods

• Remember, a class containing any abstract methods must be abstract

• It is also possible to define an abstract class that does not contain any abstract methods
  – This class is used as a base class for defining a new subclass
Superclass of abstract class may be concrete

• A subclass can be abstract even if its superclass is concrete
  – For example, the Object class is concrete, but its subclasses (e.g., GeometricObject) may be abstract
Concrete method overridden to be abstract

• A subclass can override a method from its superclass to define it abstract

• **This is rare**, but useful when the implementation of the method in the superclass becomes invalid in the subclass
  – In this case, the subclass must be defined abstract
Abstract class as a data type

• Remember, an abstract class cannot be instantiated using the new operator
• However, an abstract class can be used as a data type
  – Example
    
    ```java
    GeometricObject[] objects = new GeometricObject[2];
    objects[0] = new Circle();
    objects[1] = new Rectangle();
    ```
  

Abstract class example

- **Number** is an abstract superclass for the numeric wrapper classes (see lecture 11)
  - [https://docs.oracle.com/javase/8/docs/api/java/lang/Number.html](https://docs.oracle.com/javase/8/docs/api/java/lang/Number.html)
  - [https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/Number.html](https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/Number.html)

```
java.lang.Number
+byteValue(): byte
+shortValue(): short
+intValue(): int
+longValue(): long
+floatValue(): float
+doubleValue(): double
```

byteValue() and shortValue() call intValue() and cast result to byte and short, respectively
Abstract class example

java.util.Calendar is an abstract base class for extracting detailed information such as year, month, date, hour, minute, and second from a Date object

- An instance of java.util.Date represents a specific instant in time with millisecond precision

Subclasses of Calendar can implement specific calendar systems such as Gregorian calendar

- GregorianCalendar is a concrete subclass of the abstract class Calendar
- Developers can extend Calendar to implement others (e.g., Lunar Calendar, Jewish calendar)
Example: **GregorianCalendar** is a concrete subclass of the abstract class **Calendar**

### java.util.Calendar

- *Calendar()*
  - Constructs a default calendar.
  - Returns the value of the given calendar field.
  - Sets the given calendar to the specified value.
- *get(field: int): int*
  - Sets the calendar with the specified year, month, and date. The month parameter is 0-based; that is, 0 is for January.
  - Returns the maximum value that the specified calendar field could have.
- *set(field: int, value: int): void*
  - Adds or subtracts the specified amount of time to the given calendar field.
- *set(year: int, month: int, dayOfMonth: int): void*
  - Returns a **Date** object representing this calendar’s time value (million second offset from the UNIX epoch).
  - Sets this calendar’s time with the given **Date** object.
- *add(field: int, amount: int): void*
- *getTime(): java.util.Date*
- *setTime(date: java.util.Date): void*

### java.util.GregorianCalendar

- *GregorianCalendar()*
  - Constructs a **GregorianCalendar** for the current time.
- *GregorianCalendar(year: int, month: int, dayOfMonth: int)*
  - Constructs a **GregorianCalendar** for the specified year, month, and date.
- *GregorianCalendar(year: int, month: int, dayOfMonth: int, hour: int, minute: int, second: int)*
  - Constructs a **GregorianCalendar** for the specified year, month, date, hour, minute, and second. The month parameter is 0-based, that is, 0 is for January.
The GregorianCalendar Class

https://docs.oracle.com/javase/8/docs/api/java/util/GregorianCalendar.html
https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/GregorianCalendar.html

- Use new GregorianCalendar() to construct a default GregorianCalendar with the current time
- Use new GregorianCalendar(year, month, date) to construct a GregorianCalendar with the specified year, month, and date
  - The month parameter is 0-based (e.g., 0 is January)
Calendar is an abstract base class

- The `get(int field)` method defined in the Calendar class is useful to extract the date and time information from a Calendar object.
- The fields are defined as constants:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>YEAR</td>
<td>The year of the calendar.</td>
</tr>
<tr>
<td>MONTH</td>
<td>The month of the calendar, with 0 for January.</td>
</tr>
<tr>
<td>DATE</td>
<td>The day of the calendar.</td>
</tr>
<tr>
<td>HOUR</td>
<td>The hour of the calendar (12-hour notation).</td>
</tr>
<tr>
<td>HOUR_OF_DAY</td>
<td>The hour of the calendar (24-hour notation).</td>
</tr>
<tr>
<td>MINUTE</td>
<td>The minute of the calendar.</td>
</tr>
<tr>
<td>SECOND</td>
<td>The second of the calendar.</td>
</tr>
<tr>
<td>DAY_OF_WEEK</td>
<td>The day number within the week, with 1 for Sunday.</td>
</tr>
<tr>
<td>DAY_OF_MONTH</td>
<td>Same as DATE.</td>
</tr>
<tr>
<td>DAY_OF_YEAR</td>
<td>The day number in the year, with 1 for the first day of the year.</td>
</tr>
<tr>
<td>WEEK_OF_MONTH</td>
<td>The week number within the month, with 1 for the first week.</td>
</tr>
<tr>
<td>WEEK_OF_YEAR</td>
<td>The week number within the year, with 1 for the first week.</td>
</tr>
<tr>
<td>AM_PM</td>
<td>Indicator for AM or PM (0 for AM and 1 for PM).</td>
</tr>
</tbody>
</table>
The abstract add method

• The add method is abstract in the Calendar class because its implementation is dependent on a concrete calendar system

• add(field, value) adds the specific amount to a given field
  – Example
    • Add 7 days to the current time of the calendar
      add(Calendar.DAY_OF_MONTH, 7)
public static void main(String[] args) {
    // Construct a Gregorian calendar for the current date and time
    Calendar calendar = new GregorianCalendar();
    System.out.println("Current time is " + new Date());
    System.out.println("YEAR: " + calendar.get(Calendar.YEAR));
    System.out.println("MONTH: " + calendar.get(Calendar.MONTH));
    System.out.println("DATE: " + calendar.get(Calendar.DATE));
    System.out.println("HOUR: " + calendar.get(Calendar.HOUR));
    System.out.println("HOUR_OF_DAY: " +
                         calendar.get(Calendar.HOUR_OF_DAY));
    System.out.println("MINUTE: " + calendar.get(Calendar.MINUTE));
    System.out.println("SECOND: " + calendar.get(Calendar.SECOND));
    System.out.println("DAY_OF_WEEK: " +
                           calendar.get(Calendar.DAY_OF_WEEK));
    System.out.println("DAY_OF_MONTH: " +
                           calendar.get(Calendar.DAY_OF_MONTH));
    System.out.println("DAY_OF_YEAR: " +
                           calendar.get(Calendar.DAY_OF_YEAR));
    System.out.println("WEEK_OF_MONTH: " +
                           calendar.get(Calendar.WEEK_OF_MONTH));
    System.out.println("WEEK_OF_YEAR: " +
                           calendar.get(Calendar.WEEK_OF_YEAR));
    System.out.println("AM_PM: " + calendar.get(Calendar.AM_PM));

    // Construct a calendar for June 14, 2024
    Calendar calendar1 = new GregorianCalendar(2024, 5, 14);
    String[] dayNameOfWeek = {"Sunday", "Monday", "Tuesday", "Wednesday",
                               "Thursday", "Friday", "Saturday"};
    System.out.println("June 14, 2024, is a " +
                           dayNameOfWeek[calendar1.get(Calendar.DAY_OF_WEEK) - 1]);
}
Next Lecture

• Interfaces