Abstract Classes

Introduction to Programming and Computational Problem Solving II

CSE 8B

Lecture 15

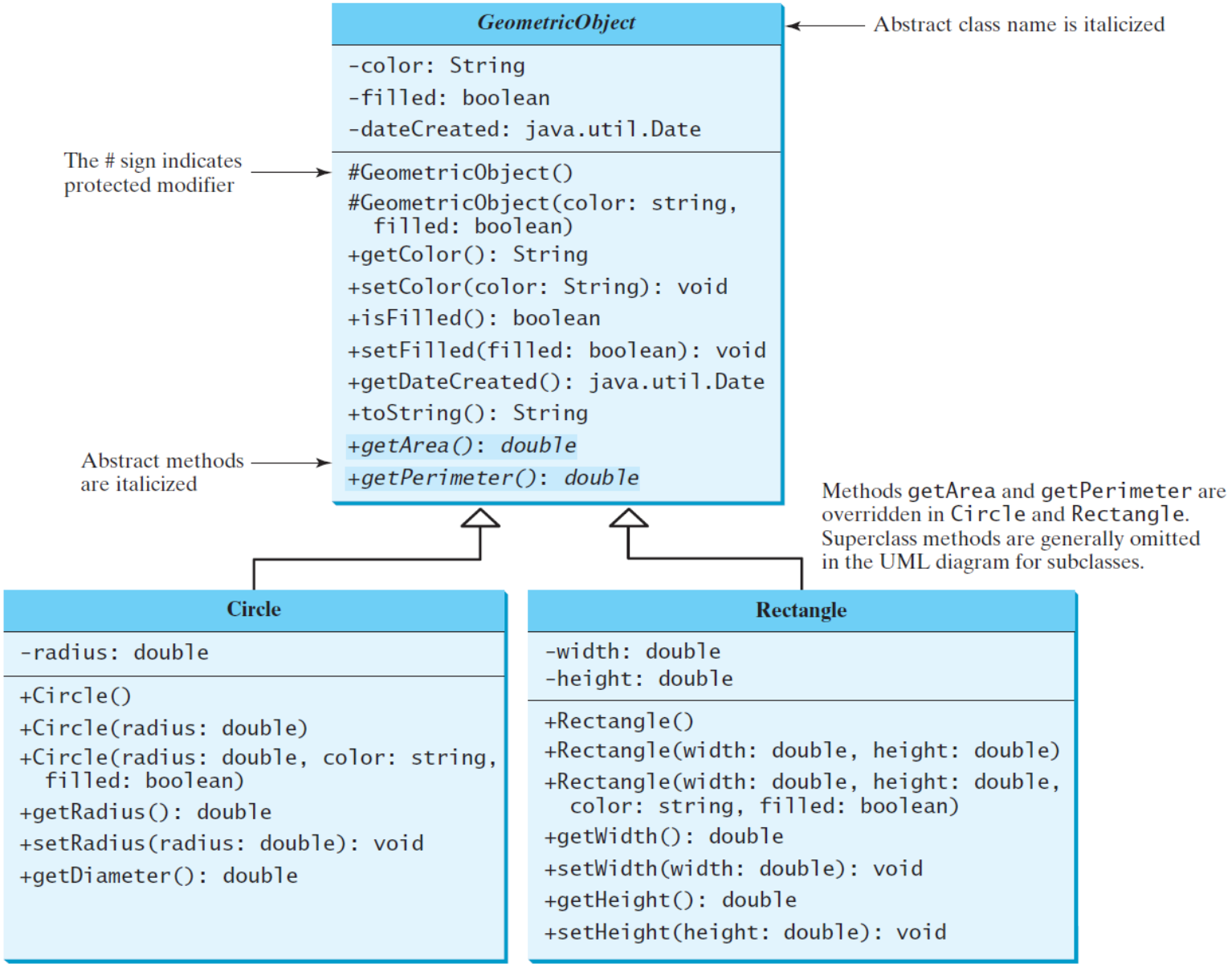
Announcements

- Assignment 6 is due Mar 5, 11:59 PM
  - Upgrade beginning Mar 8, 12:01 AM
- Assignment 7 will be released Mar 5
  - Due Mar 12, 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

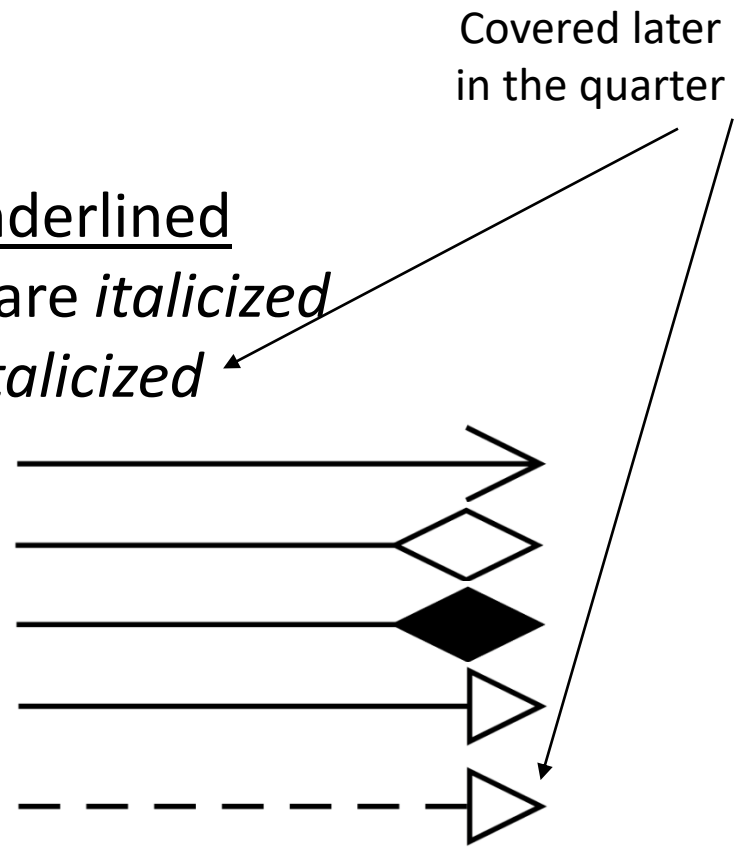
Abstract class example



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



Methods and data fields visibility

Modifiers on Members in a Class	Accessed from Same Class	Accessed from Any Class in Same Package	Accessed from Any Class in Same Package and Any Subclass in Any Package	Accessed from Any Class in Any Package
Public	✓	✓	✓	✓
Protected	✓	✓	✓	
Default (no modifier)	✓	✓		
Private	✓			

abstract modifier

- Abstract classes and abstract methods are denoted using the abstract modifier
  - Example

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

  // Construct a default geometric object
  protected GeometricObject() {
    dateCreated = new java.util.Date();
  }

  // Construct a geometric object with color and filled value
  protected GeometricObject(String color, boolean filled) {
    dateCreated = new java.util.Date();
    this.color = color;
    this.filled = filled;
  }

  ...

  // Abstract method getArea
  public abstract double getArea();

  // Abstract method getPerimeter
  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

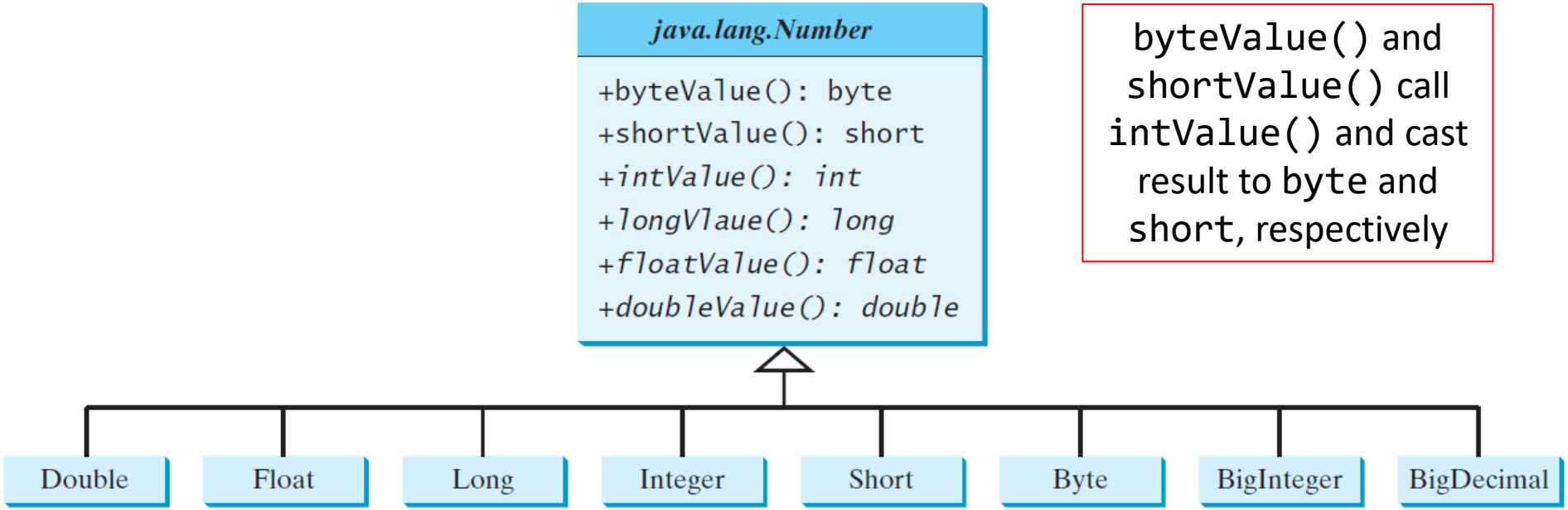
Abstract class as a data type

- Remember, an abstract class cannot be instantiated using the new operator
- However, an abstract class can be used as a data type
  - Example

```
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/en/java/javase/11/docs/api/java.base/java/lang/Number.html



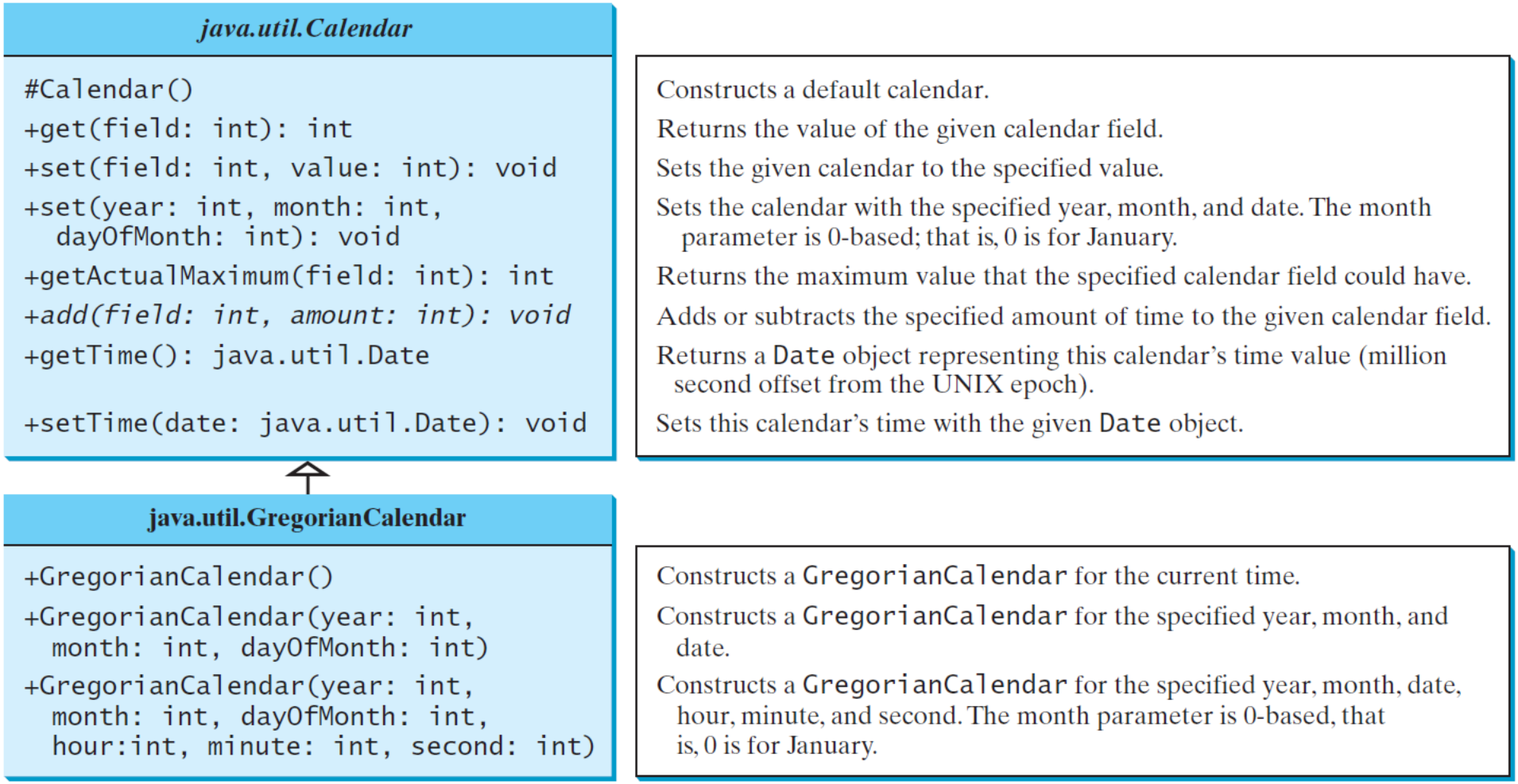
`byteValue()` and `shortValue()` call `intValue()` and cast result to `byte` and `short`, respectively

Abstract class example

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

- `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`



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

Constant	*Description*
`YEAR`	The year of the calendar.
`MONTH`	The month of the calendar, with 0 for January.
`DATE`	The day of the calendar.
`HOUR`	The hour of the calendar (12-hour notation).
`HOUR_OF_DAY`	The hour of the calendar (24-hour notation).
`MINUTE`	The minute of the calendar.
`SECOND`	The second of the calendar.
`DAY_OF_WEEK`	The day number within the week, with 1 for Sunday.
`DAY_OF_MONTH`	Same as DATE.
`DAY_OF_YEAR`	The day number in the year, with 1 for the first day of the year.
`WEEK_OF_MONTH`	The week number within the month, with 1 for the first week.
`WEEK_OF_YEAR`	The week number within the year, with 1 for the first week.
`AM_PM`	Indicator for AM or PM (0 for AM and 1 for PM).

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)
```

Getting date/time information

```
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 December 14, 2024
    Calendar calendar1 = new GregorianCalendar(2024, 11, 14);
    String[] dayNameOfWeek = {"Sunday", "Monday", "Tuesday", "Wednesday",
        "Thursday", "Friday", "Saturday"};
    System.out.println("December 14, 2024, is a " +
      dayNameOfWeek[calendar1.get(Calendar.DAY_OF_WEEK) - 1]);
  }
```

Next Lecture

- Interfaces