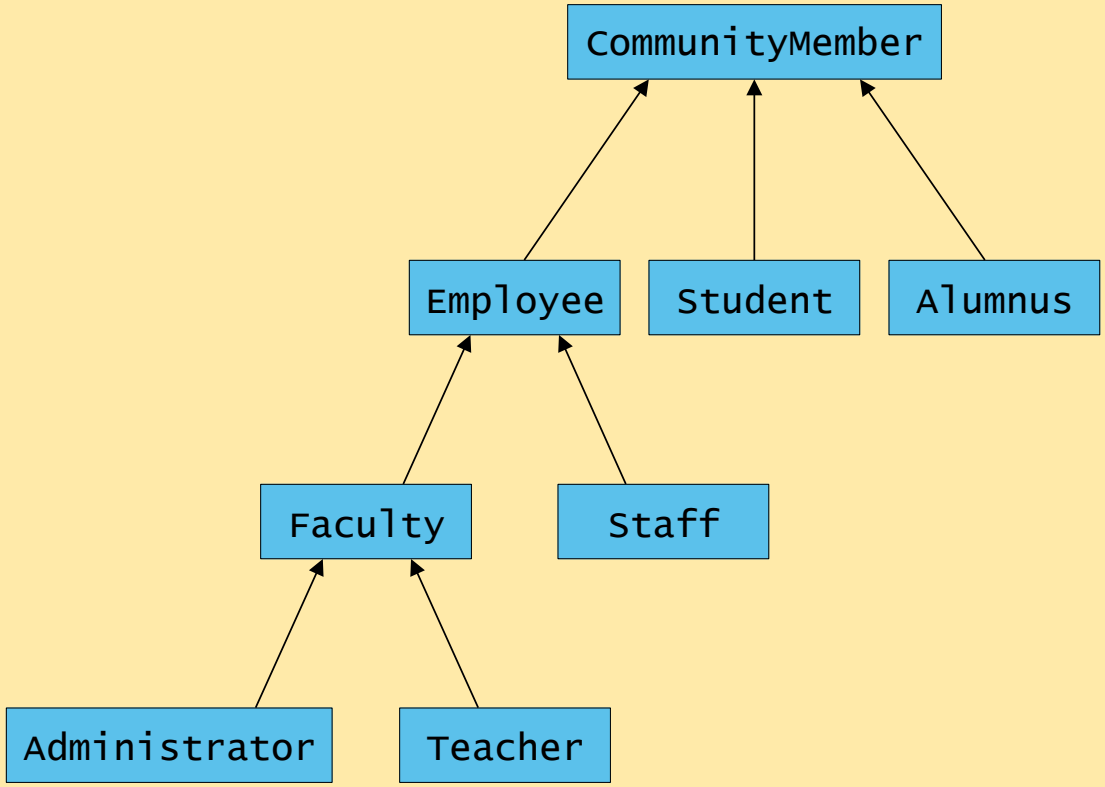
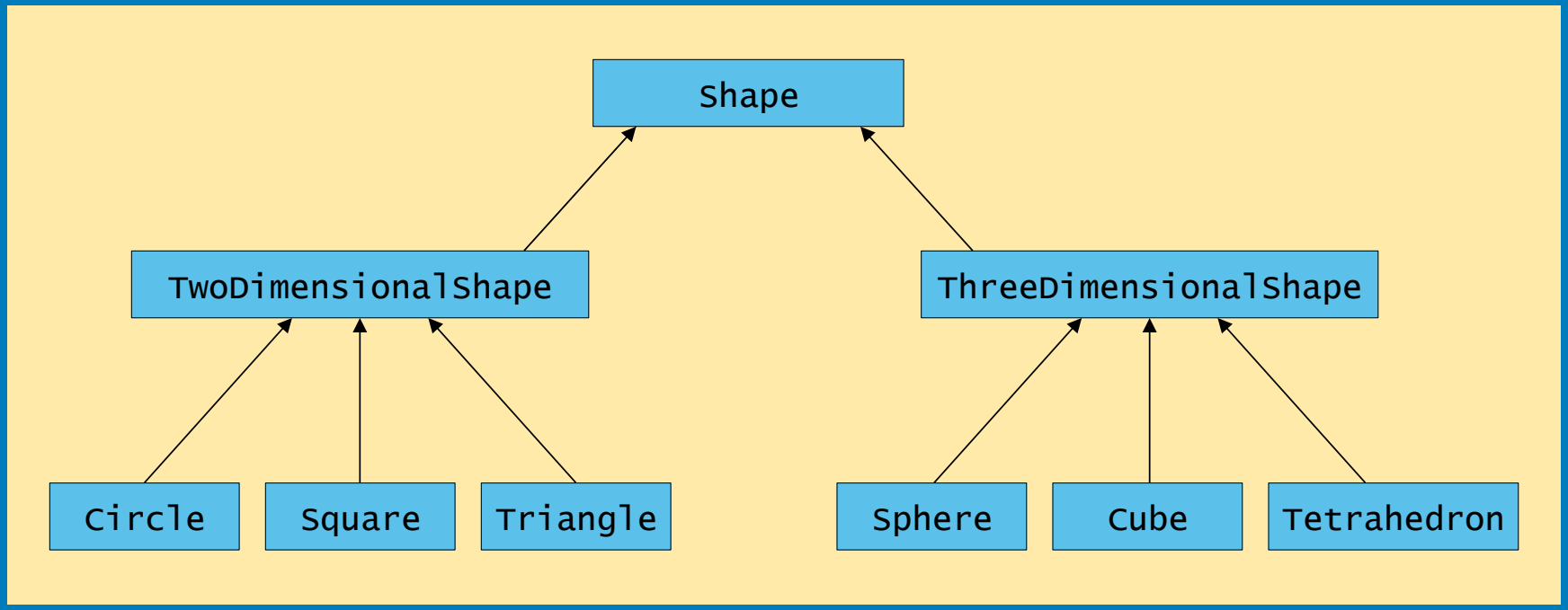
Object-Oriented Programming
Inheritance

Superclasses and Subclasses

- Superclasses and subclasses
  - Object of one class "is an" object of another class
    - Example: Rectangle is quadrilateral.
      - Class `Rectangle` inherits from class `Quadrilateral`
      - `Quadrilateral`: superclass
      - `Rectangle`: subclass
  - Superclass typically represents larger set of objects than subclasses
    - Example:
      - superclass: `Vehicle`
        - Cars, trucks, boats, bicycles, …
      - subclass: `Car`
        - Smaller, more-specific subset of vehicles



Inheritance hierarchy for university `CommunityMember`s.



Inheritance hierarchy for Shapes.

Superclasses and Subclasses (Cont.)

- Inheritance examples

Superclass	Subclasses
`Student`	`GraduateStudent`, `UndergraduateStudent`
`Shape`	`Circle`, `Triangle`, `Rectangle`
`Loan`	`CarLoan`, `HomeImprovementLoan`, `MortgageLoan`
`Employee`	`Faculty`, `Staff`
`BankAccount`	`CheckingAccount`, `SavingsAccount`

Inheritance examples.

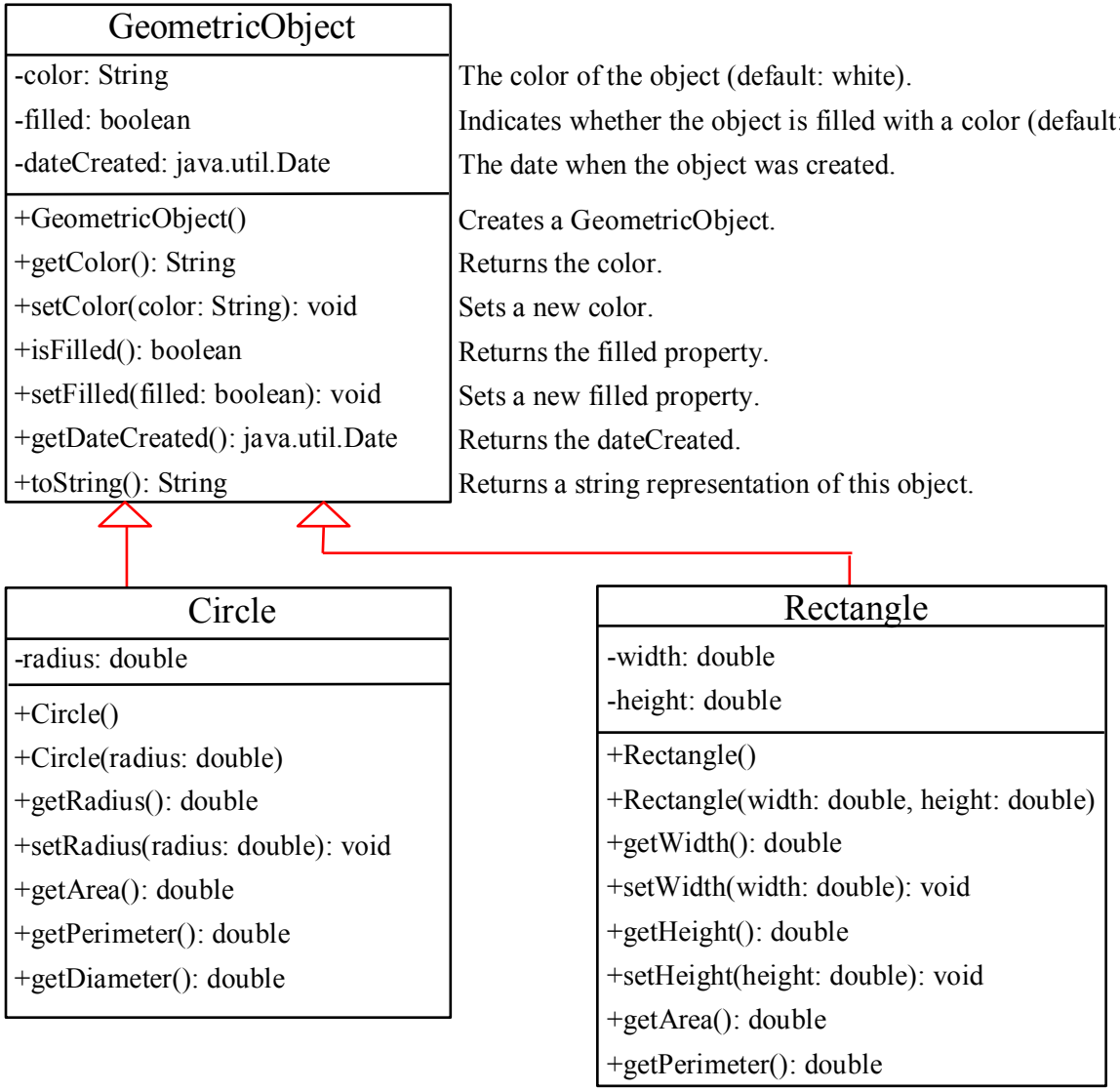
Superclasses and Subclasses (Cont.)

- Inheritance hierarchy
  - Inheritance relationships: tree-like hierarchy structure
  - Each class becomes
    - superclass
      - Supply data/behaviors to other classes

    OR

    - subclass
      - Inherit data/behaviors from other classes

Superclasses and Subclasses



GeometricObject

Circle

Rectangle

TestCircleRectangle

Are superclass's Constructor Inherited?

No. They are not inherited.

They are invoked explicitly or implicitly.

Explicitly using the ***super*** keyword.

A constructor is used to construct an instance of a class. Unlike properties and methods, a superclass's constructors are not inherited in the subclass. They can only be invoked from the subclasses' constructors, using the keyword super. *If the keyword super is not explicitly used, the superclass's no-arg constructor is automatically invoked.*

Superclass's Constructor Is Always Invoked

A constructor may invoke an overloaded constructor or its superclass's constructor. If none of them is invoked explicitly, the compiler puts super() as the first statement in the constructor. For example,

```
public A() {
}
```

is equivalent to

```
public A() {
  super();
}
```

```
public A(double d) {
  // some statements
}
```

is equivalent to

```
public A(double d) {
  super();
  // some statements
}
```

Using the Keyword `super`

The keyword `super` refers to the superclass of the class in which `super` appears. This keyword can be used in two ways:

- To call a superclass constructor
- To call a superclass method

CAUTION

You must use the keyword super to call the superclass constructor. Invoking a superclass constructor's name in a subclass causes a syntax error. Java requires that the statement that uses the keyword super appear first in the constructor.

Constructor Chaining

Constructing an instance of a class invokes all the superclasses' constructors along the inheritance chain. This is called *constructor chaining*.

```
public class Faculty extends Employee {
  public static void main(String[] args) {
    new Faculty();
  }

  public Faculty() {
    System.out.println("(4) Faculty's no-arg constructor is invoked");
  }
}

class Employee extends Person {
  public Employee() {
    this("(2) Invoke Employee's overloaded constructor");
    System.out.println("(3) Employee's no-arg constructor is invoked");
  }

  public Employee(String s) {
    System.out.println(s);
  }
}

class Person {
  public Person() {
    System.out.println("(1) Person's no-arg constructor is invoked");
  }
}
```

Trace Execution

```
public class Faculty extends Employee {
  public static void main(String[] args) {
    new Faculty();
  }

  public Faculty() {
    System.out.println("(4) Faculty's no-arg constructor is invoked");
  }
}

class Employee extends Person {
  public Employee() {
    this("(2) Invoke Employee's overloaded constructor");
    System.out.println("(3) Employee's no-arg constructor is invoked");
  }

  public Employee(String s) {
    System.out.println(s);
  }
}

class Person {
  public Person() {
    System.out.println("(1) Person's no-arg constructor is invoked");
  }
}
```

1. Start from the main method

Trace Execution

```
public class Faculty extends Employee {
  public static void main(String[] args) {
    new Faculty();
  }

  public Faculty() {
    System.out.println("(4) Faculty's no-arg constructor is invoked");
  }
}

class Employee extends Person {
  public Employee() {
    this("(2) Invoke Employee's overloaded constructor");
    System.out.println("(3) Employee's no-arg constructor is invoked");
  }

  public Employee(String s) {
    System.out.println(s);
  }
}

class Person {
  public Person() {
    System.out.println("(1) Person's no-arg constructor is invoked");
  }
}
```

2. Invoke Faculty constructor

Trace Execution

```
public class Faculty extends Employee {
  public static void main(String[] args) {
    new Faculty();
  }

  public Faculty() {
    System.out.println("(4) Faculty's no-arg constructor is invoked");
  }
}

class Employee extends Person {
  public Employee() {
    this("(2) Invoke Employee's overloaded constructor");
    System.out.println("(3) Employee's no-arg constructor is invoked");
  }

  public Employee(String s) {
    System.out.println(s);
  }
}

class Person {
  public Person() {
    System.out.println("(1) Person's no-arg constructor is invoked");
  }
}
```

3. Invoke Employee's no-arg constructor

Trace Execution

```
public class Faculty extends Employee {
  public static void main(String[] args) {
    new Faculty();
  }

  public Faculty() {
    System.out.println("(4) Faculty's no-arg constructor is invoked");
  }
}

class Employee extends Person {
  public Employee() {
    this("(2) Invoke Employee's overloaded constructor");
    System.out.println("(3) Employee's no-arg constructor is invoked");
  }

  public Employee(String s) {
    System.out.println(s);
  }
}

class Person {
  public Person() {
    System.out.println("(1) Person's no-arg constructor is invoked");
  }
}
```

4. Invoke Employee(String) constructor

Trace Execution

```
public class Faculty extends Employee {
  public static void main(String[] args) {
    new Faculty();
  }

  public Faculty() {
    System.out.println("(4) Faculty's no-arg constructor is invoked");
  }
}

class Employee extends Person {
  public Employee() {
    this("(2) Invoke Employee's overloaded constructor");
    System.out.println("(3) Employee's no-arg constructor is invoked");
  }

  public Employee(String s) {
    System.out.println(s);
  }
}

class Person {
  public Person() {
    System.out.println("(1) Person's no-arg constructor is invoked");
  }
}
```

5. Invoke Person() constructor

Trace Execution

```
public class Faculty extends Employee {
  public static void main(String[] args) {
    new Faculty();
  }

  public Faculty() {
    System.out.println("(4) Faculty's no-arg constructor is invoked");
  }
}

class Employee extends Person {
  public Employee() {
    this("(2) Invoke Employee's overloaded constructor");
    System.out.println("(3) Employee's no-arg constructor is invoked");
  }

  public Employee(String s) {
    System.out.println(s);
  }
}

class Person {
  public Person() {
    System.out.println("(1) Person's no-arg constructor is invoked");
  }
}
```

6. Execute println

Trace Execution

```
public class Faculty extends Employee {
  public static void main(String[] args) {
    new Faculty();
  }

  public Faculty() {
    System.out.println("(4) Faculty's no-arg constructor is invoked");
  }
}

class Employee extends Person {
  public Employee() {
    this("(2) Invoke Employee's overloaded constructor");
    System.out.println("(3) Employee's no-arg constructor is invoked");
  }

  public Employee(String s) {
    System.out.println(s);
  }
}

class Person {
  public Person() {
    System.out.println("(1) Person's no-arg constructor is invoked");
  }
}
```

7. Execute println

Trace Execution

```
public class Faculty extends Employee {
  public static void main(String[] args) {
    new Faculty();
  }

  public Faculty() {
    System.out.println("(4) Faculty's no-arg constructor is invoked");
  }
}

class Employee extends Person {
  public Employee() {
    this("(2) Invoke Employee's overloaded constructor");
    System.out.println("(3) Employee's no-arg constructor is invoked");
  }

  public Employee(String s) {
    System.out.println(s);
  }
}

class Person {
  public Person() {
    System.out.println("(1) Person's no-arg constructor is invoked");
  }
}
```

8. Execute println

Trace Execution

```
public class Faculty extends Employee {
  public static void main(String[] args) {
    new Faculty();
  }

  public Faculty() {
    System.out.println("(4) Faculty's no-arg constructor is invoked");
  }
}

class Employee extends Person {
  public Employee() {
    this("(2) Invoke Employee's overloaded constructor");
    System.out.println("(3) Employee's no-arg constructor is invoked");
  }

  public Employee(String s) {
    System.out.println(s);
  }
}

class Person {
  public Person() {
    System.out.println("(1) Person's no-arg constructor is invoked");
  }
}
```

9. Execute println

Example on the Impact of a Superclass without no-arg Constructor

Find out the errors in the program:

```
public class Apple extends Fruit {
}

class Fruit {
  public Fruit(String name) {
    System.out.println("Fruit's constructor is invoked");
  }
}
```

Declaring a Subclass

A subclass extends properties and methods from the superclass. You can also:

- Add new properties (Seen already)
- Add new methods (Seen already)
- Override the methods of the superclass (to be discussed)

Calling Superclass Methods

You could rewrite the printCircle() method in the Circle class as follows:

```
public void printCircle() {
  System.out.println("The circle is created " +
    super.getDateCreated() + " and the radius is " + radius);
}
```

Overriding Methods in the Superclass

A subclass inherits methods from a superclass. Sometimes it is necessary for the subclass to modify the implementation of a method defined in the superclass. This is referred to as *method overriding*.

```
public class Circle extends GeometricObject {
  // Other methods are omitted


  /** Override the toString method defined in GeometricObject */
  public String toString() {
    return super.toString() + "\nradius is " + radius;
  }
}
```

NOTE

An instance method can be overridden only if it is accessible. Thus a private method cannot be overridden, because it is not accessible outside its own class. If a method defined in a subclass is private in its superclass, the two methods are completely unrelated.

NOTE

Like an instance method, a static method can be inherited. However, a static method cannot be overridden. If a static method defined in the superclass is redefined in a subclass, the method defined in the superclass is hidden.

Overriding vs. Overloading

```
public class Test {
  public static void main(String[] args) {
    A a = new A();
    a.p(10);
  }
}

class B {
  public void p(int i) {
  }
}

class A extends B {
  // This method overrides the method in B
  public void p(int i) {
    System.out.println(i);
  }
}
```

```
public class Test {
  public static void main(String[] args) {
    A a = new A();
    a.p(10);
  }
}

class B {
  public void p(int i) {
  }
}

class A extends B {
  // This method overloads the method in B
  public void p(double i) {
    System.out.println(i);
  }
}
```

The `Object` Class

Every class in Java is descended from the java.lang.Object class. If no inheritance is specified when a class is defined, the superclass of the class is Object.

```
public class Circle {
  ...
}
```

Equivalent

```
public class Circle extends Object {
  ...
}
```

The toString() method in Object

The toString() method returns a string representation of the object. The default implementation returns a string consisting of a class name of which the object is an instance, the at sign (@), and a number representing this object.

```
Loan loan = new Loan();
System.out.println(loan.toString());
```

The code displays something like Loan@15037e5 . This message is not very helpful or informative. Usually you should override the toString method so that it returns a digestible string representation of the object.