Inheritance, Polymorphism, Overloading, and Overriding

Object Oriented Programming

```
public class Person {
   String name;
   char gender;
   Date birthday;

   int getAge(Date today) {
      …
   }
}
```

```
public class Student
      extends Person {

   Vector<Grade> grades;

   double getGPA() {
      …
   }
}
```

```
public class Professor
      extends Person {

   Vector<Paper> papers;

   int getCiteCount() {
      …
   }
}
```

Inheritance

- “is-a” relationship
- Single inheritance:
  - Subclass is derived from one existing class (superclass)
- Multiple inheritance:
  - Subclass is derived from more than one superclass
  - Not supported by Java
  - A class can only extend the definition of one class

Main Key notes of Inheritance

- **superclass**: Parent class being extended.
- **subclass**: Child class that inherits behavior from superclass.
  - gets a copy of every field and method from superclass
- **is-a relationship**: Each object of the subclass also "is a(n)" object of the superclass and can be treated as one.

Inheritance (continued)

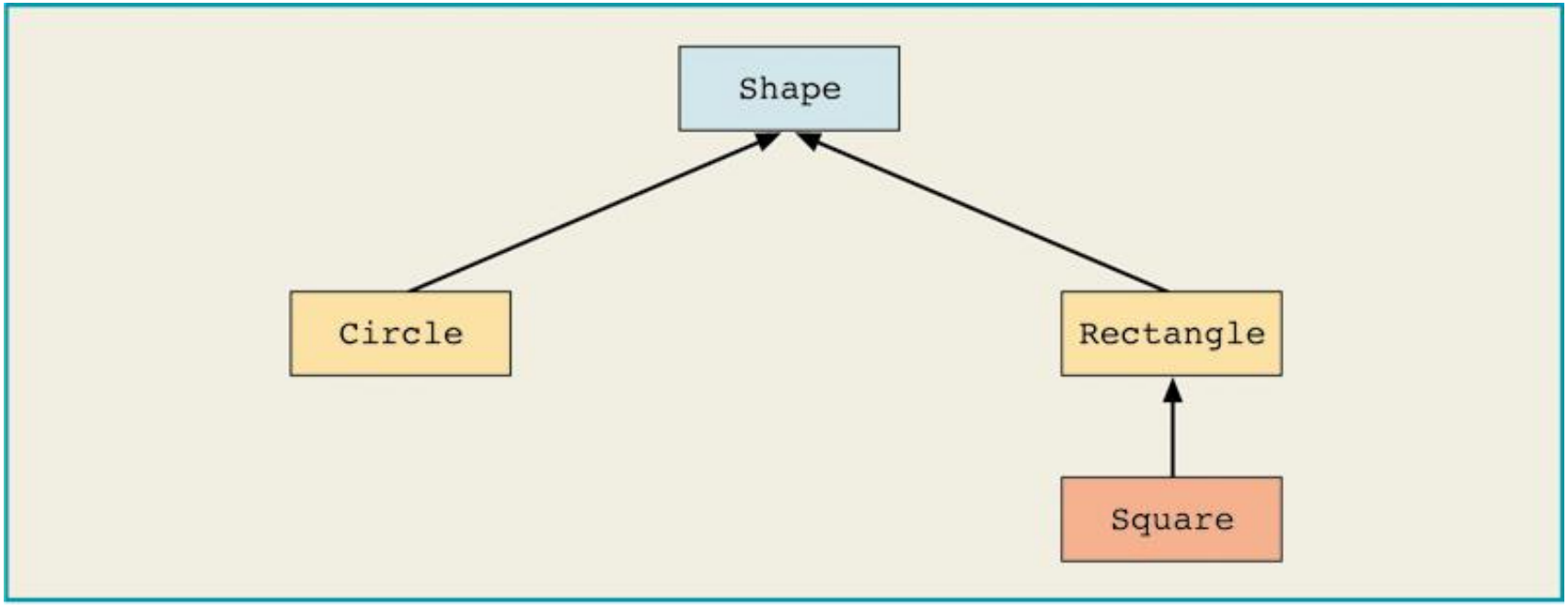

Figure 11-1 Inheritance hierarchy

Inheritance: `class Circle` Derived from `class Shape`

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

Inheritance

- Allow us to specify *relationships between types*
  - Abstraction, generalization, specification
  - The “is-a” relationship
  - Examples?
- Why is this useful in programming?
  - Allows for code reuse
  - More intuitive/expressive code

Code Reuse

- General functionality can be written once and applied to **any** subclass
- Subclasses can specialize by adding members and methods, or overriding functions

Inheritance: Adding Functionality

- Subclasses have ***all*** of the data members and methods of the superclass
- Subclasses can add to the superclass
  - Additional data members
  - Additional methods
- Subclasses are more specific and have more functionality
- Superclasses capture generic functionality common across many types of objects

```
public class Person {
   String name;
   char gender;
   Date birthday;

   int getAge(Date today) {
      …
   }
}
```

```
public class Student
      extends Person {

   Vector<Grade> grades;

   double getGPA() {
      …
   }
}
```

```
public class Professor
      extends Person {

   Vector<Paper> papers;

   int getCiteCount() {
      …
   }
}
```

Calling methods of the superclass

- To write a method's definition of a subclass, specify a call to the public method of the superclass
  - If subclass overrides public method of superclass, specify call to public method of superclass:

    `super.MethodName(parameter list)`
  - If subclass does not override public method of superclass, specify call to public method of superclass:

    `MethodName(parameter list)`

class Box

```
public void setDimension(double l, double w, double h)
{
     super.setDimension(l, w);
     if (h >= 0)
          height = h;
     else
          height = 0;
}}
```

Box overloads the method setDimension
(Different parameters)

Defining Constructors of the Subclass

- Call to constructor of superclass:
  - Must be first statement
  - Specified by super parameter list

```
public Box()
{
    super();
    height = 0;
}

public Box(double l, double w, double h)
{
    super(l, w);
    height = h;
}
```

Access Control

- Access control keywords define which classes can access classes, methods, and members

Modifier	Class	Package	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	N
none	Y	Y	N	N
private	Y	N	N	N

Polymorphism

- Can treat an object of a subclass as an object of its superclass
  - A reference variable of a superclass type can point to an object of its subclass

```
Person name, nameRef;
PartTimeEmployee employee, employeeRef;
name = new Person("John", "Blair");
employee = new PartTimeEmployee("Susan", "Johnson",
                                12.50, 45);
nameRef = employee;
System.out.println("nameRef: " + nameRef);


nameRef: Susan Johnson wages are: $562.5
```

Polymorphism

- Late binding or dynamic binding (run-time binding):
  - Method to be executed is determined at execution time, not compile time
- Polymorphism: to assign multiple meanings to the same method name
- Implemented using late binding

Polymorphism (continued)

- The reference variable `name` or `nameRef` can point to any object of the `class Person` or the `class PartTimeEmployee`
- These reference variables have many forms, that is, they are polymorphic reference variables
- They can refer to objects of their own class or to objects of the classes inherited from their class

Casting

- You cannot automatically make reference variable of subclass type point to object of its superclass
- Suppose that `supRef` is a reference variable of a superclass type and `supRef` points to an object of its subclass:
  - Can use a cast operator on `supRef` and make a reference variable of the subclass point to the object
  - If `supRef` does not point to a subclass object and you use a cast operator on `supRef` to make a reference variable of the subclass point to the object, then Java will throw a `ClassCastException`—indicating that the class cast is not allowed

Polymorphism (continued)

- Operator `instanceof`: determines whether a reference variable that points to an object is of a particular class type
- This expression evaluates to `true` if `p` points to an object of the `class` `BoxShape`; otherwise it evaluates to `false`:

```
p instanceof BoxShape
```

Polymorphism (continued)

- Can also declare a `class` final using the keyword `final`
- If a `class` is declared `final`, then no other class can be derived from this class
- Java does not use late binding for methods that are `private`, marked `final`, or `static`
  - `Why not?`

final Methods

- Can declare a method of a class final using the keyword `final`

```
public final void doSomeThing()
{
      //...
}
```

- If a method of a `class` is declared `final`, it cannot be overridden with a new definition in a derived class

Overloading Method

- Method Overloading is a feature that allows a class to have more than one method having the same name, if their argument lists are different. It is similar to constructor overloading in Java, that allows a class to have more than one constructor having different argument lists.
- Three way of overload method
  - Number of parameters e.g. add (int , int)
  - Data type of parameters
  - Sequence of Data type of parameters

Overriding Methods

- A subclass can override (redefine) the methods of the superclass
  - Objects of the subclass type will use the new method
  - Objects of the superclass type will use the original

Overriding methods

- any method that is not `final` may be overridden by a descendant class
- same signature as method in ancestor
- may not reduce visibility
- may use the original method if simply want to add more behavior to existing

I/O and exceptions

- **exception**: An object representing an error.
  - **checked exception**: One that must be handled for the program to compile.
- Many I/O tasks throw exceptions.
- When you perform I/O, you must either:
  - also **throw** that exception yourself
  - **catch** (handle) the exception

Throwing an exception

```
public type name(params) throws type {
```

- **`throws` clause**: Keywords on a method's header that state that it may generate an exception.

  "I hereby announce that this method might throw an exception, and I accept the consequences if it happens."

Catching an exception

```
try {
    statement(s);
} catch (type name) {
    code to handle the exception
}
```

- The `try` code executes. If the given exception occurs, the `try` block stops running; it jumps to the `catch` block and runs that.