AP Computer Science B

Java Object-Oriented Programming [Ver. 3.0]

Unit 4: Object-Oriented Programming

CHAPTER 12A: INHERITANCE

DR. ERIC CHOU

IEEE SENIOR MEMBER

Objectives

- Inheritance: is_A relationship
- Design of a sub-class
- Class reference: super keyword
- Example: Rectangle, Square (Reduction of Data Field), Prism (Addition of Data Field)
- Multiple-Inheritance: GeometricObject, Shape, Color, Fill
- Reference, Object, and Inheritance (Type Casting)

Overview of Inheritance

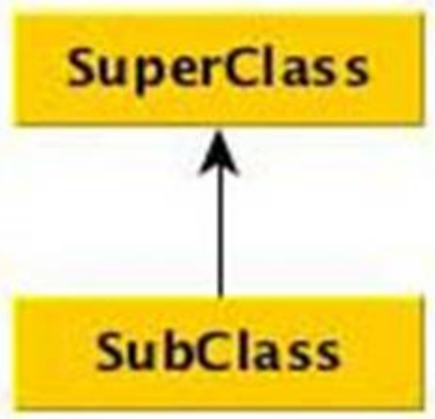
LECTURE 1

Inheritance

- Inheritance in Java begins with the relationship between two classes defined like this:

```
class SubClass extends SuperClass
```

- Inheritance expresses the is a relationship in that SubClass is a (**specialization** of) SuperClass.

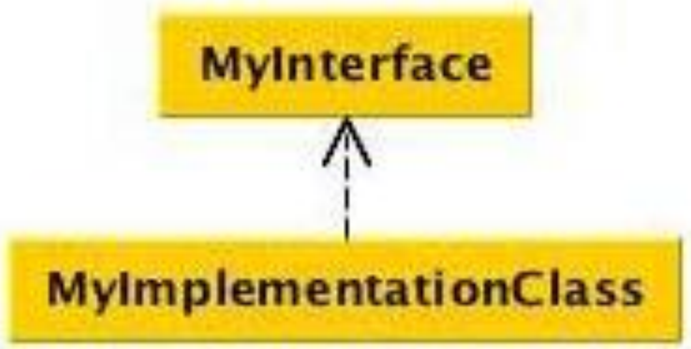


Inheritance

- The extends relation has many of the same characteristics of the implements relationship used for interfaces

```
class MyImplementationClass implements MyInterface
```

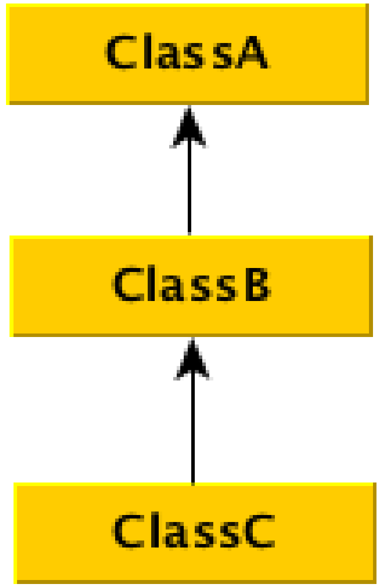
- As with inheritance, we say that **MyImplementationClass** is a **MyInterface**.



Inheritance

Diagrammatically, these relationships are expressed in UML with the extends as a solid line (white Triangle in some tools) and implements as a dashed line:

The ***is a*** relationship is transitive in that, if we have this hierarchy:



Inheritance

in which

ClassB is a ClassA

ClassC is a ClassB

then, by transitivity:

ClassC is a ClassA

The term base class is also used for superclass, and derived class as subclass.

Being a subclass is also transitive in that we can say that:

ClassC is a subclass of ClassA

The term inheritance expresses the fact that the objects of the subclass inherit all the features of the superclass including data members and functions, although the private data members and functions of the superclass are **not** directly accessible.

What does a subclass inherit?

- A **subclass inherits** all the members (fields, methods, and nested classes) from its superclass. **Constructors** are not members, so they are not **inherited** by **subclasses**, but the constructor of the superclass can be invoked from the **subclass**.
- Members of a class that are declared private are not directly accessible by subclasses of that class. Only members of a class that are declared **protected** or **public** are accessed directly by subclasses declared in a package other than the one in which the class is declared.

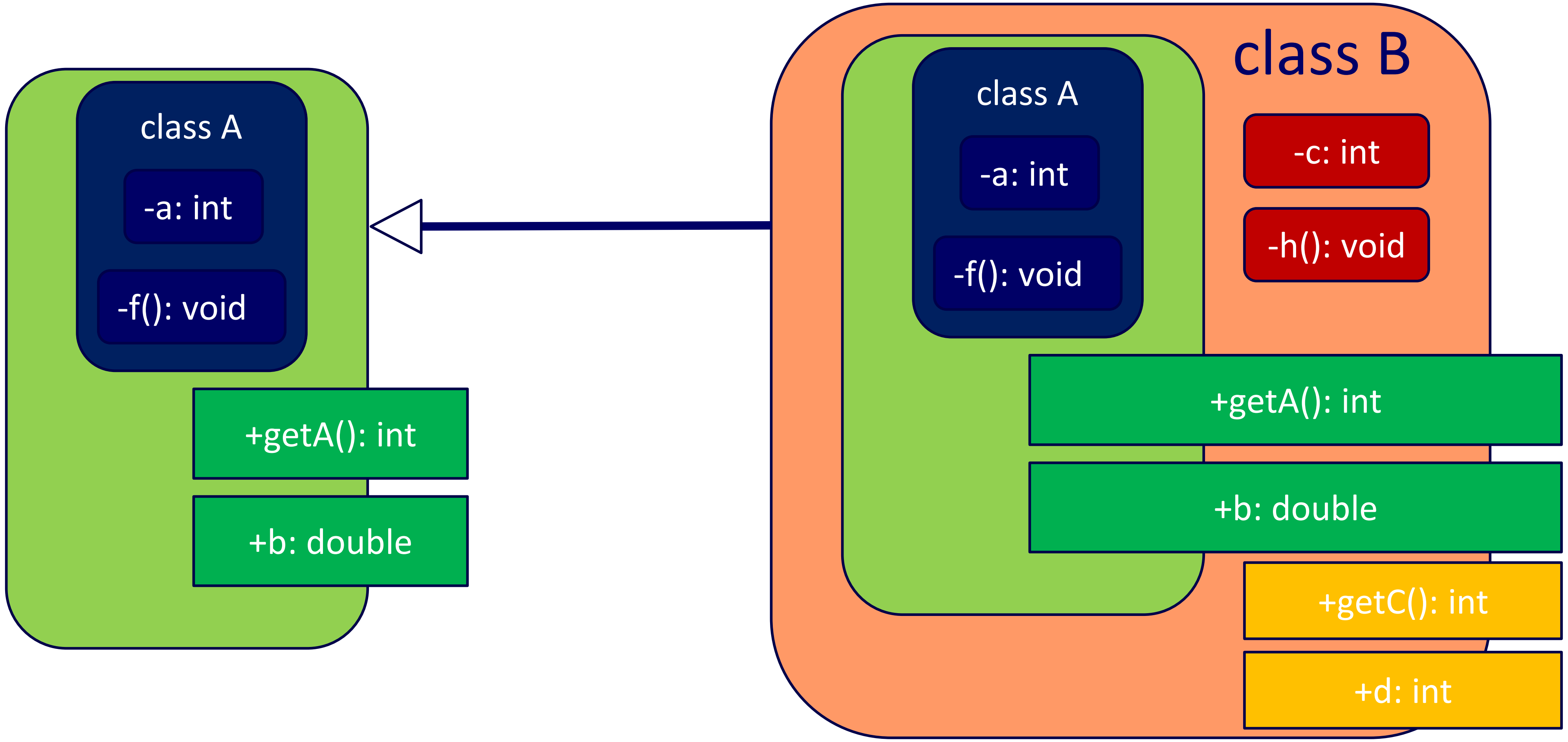
protected: no access by other package but can be inherited.

Visibility Modifiers

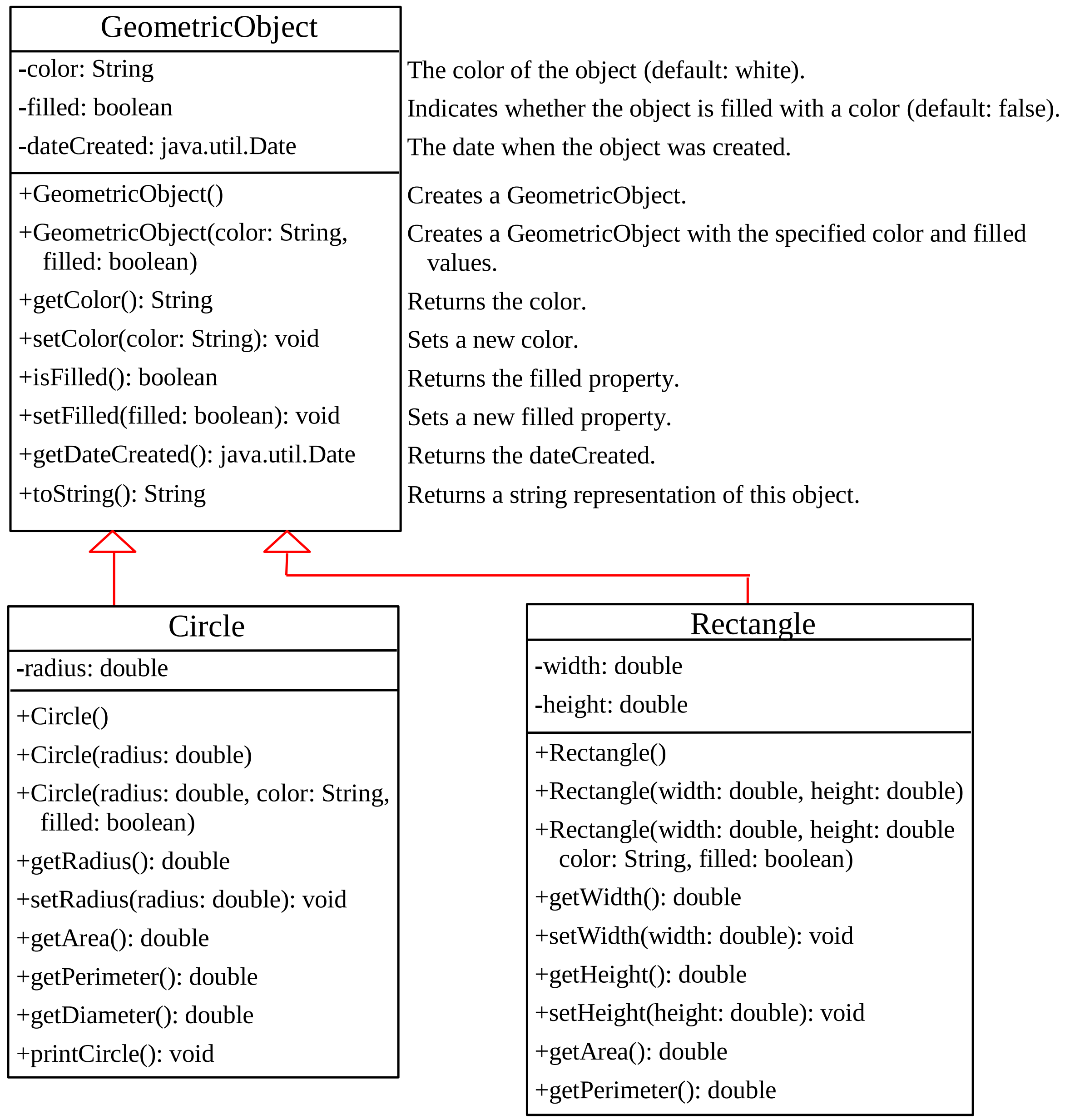
Modifier	Inheritance	Access
+public	All	All
#protected	All	Subclasses
~default (none)	All	Same package
-private	All	Same class

Accessing an Objects of subclasses

(Subclass is another kind of Wrapper Class)



Superclasses and Subclasses



GeometricObject	
-color: String	The color of the object (default: white).
-filled: boolean	Indicates whether the object is filled with a color (default: false).
-dateCreated: java.util.Date	The date when the object was created.
+GeometricObject()	Creates a GeometricObject.
+GeometricObject(color: String, filled: boolean)	Creates a GeometricObject with the specified color and filled values.
+getColor(): String	Returns the color.
+setColor(color: String): void	Sets a new color.
+isFilled(): boolean	Returns the filled property.
+setFilled(filled: boolean): void	Sets a new filled property.
+getDateCreated(): java.util.Date	Returns the dateCreated.
+toString(): String	Returns a string representation of this object.

Circle
-radius: double
+Circle()
+Circle(radius: double)
+Circle(radius: double, color: String, filled: boolean)
+getRadius(): double
+setRadius(radius: double): void
+getArea(): double
+getPerimeter(): double
+getDiameter(): double
+printCircle(): void

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
+getArea(): double
+getPerimeter(): double

Demonstration Program

TESTCIRCLERECTANGLE.JAVA
GEOMETRICOBJECT.JAVA
CIRCLEFROMSIMPLEGEOMETRICOBJECT.JAVA
RECTANGLEFROMSIMPLEGEOMETRICOBJECT.JAVA

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();
}
```

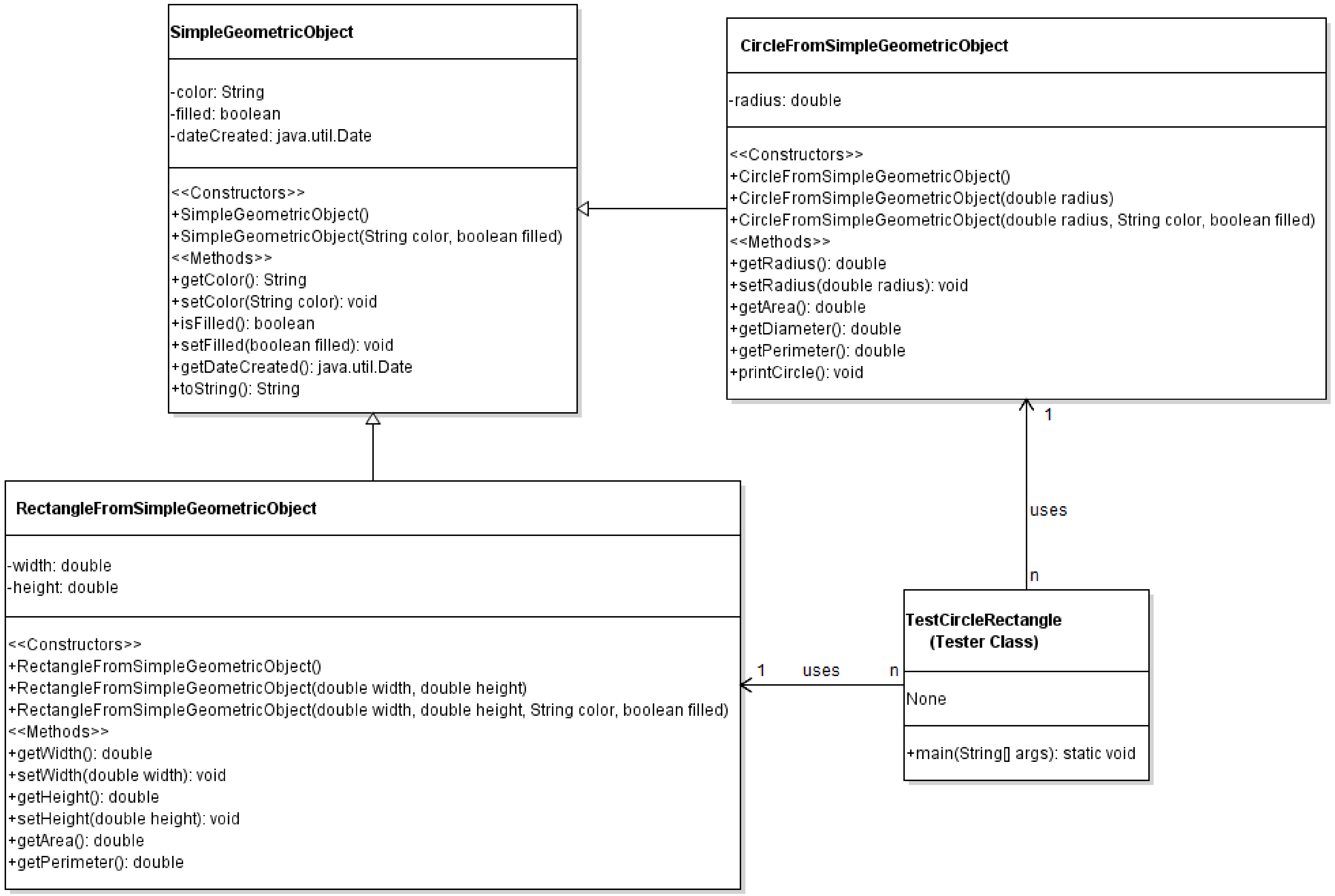
```
public A(double d) {
  // some statements
}
```

is equivalent to

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

Demonstration Program

VIOLET UML FOR INHERITANCE


SimpleGeometricObject
-color: String
-filled: boolean
-dateCreated: java.util.Date
<<Constructors>>
+SimpleGeometricObject()
+SimpleGeometricObject(String color, boolean filled)
<<Methods>>
+getColor(): String
+setColor(String color): void
+isFilled(): boolean
+setFilled(boolean filled): void
+getDateCreated(): java.util.Date
+toString(): String
CircleFromSimpleGeometricObject
-radius: double
<<Constructors>>
+CircleFromSimpleGeometricObject()
+CircleFromSimpleGeometricObject(double radius)
+CircleFromSimpleGeometricObject(double radius, String color, boolean filled)
<<Methods>>
+getRadius(): double
+setRadius(double radius): void
+getArea(): double
+getDiameter(): double
+getPerimeter(): double
+printCircle(): void
1
uses
n
RectangleFromSimpleGeometricObject
-width: double
-height: double
<<Constructors>>
+RectangleFromSimpleGeometricObject()
+RectangleFromSimpleGeometricObject(double width, double height)
+RectangleFromSimpleGeometricObject(double width, double height, String color, boolean filled)
<<Methods>>
+getWidth(): double
+setWidth(double width): void
+getHeight(): double
+setHeight(double height): void
+getArea(): double
+getPerimeter(): double
1
uses
n
TestCircleRectangle
(Tester Class)
None
+main(String[] args): static void

Design of a sub-class

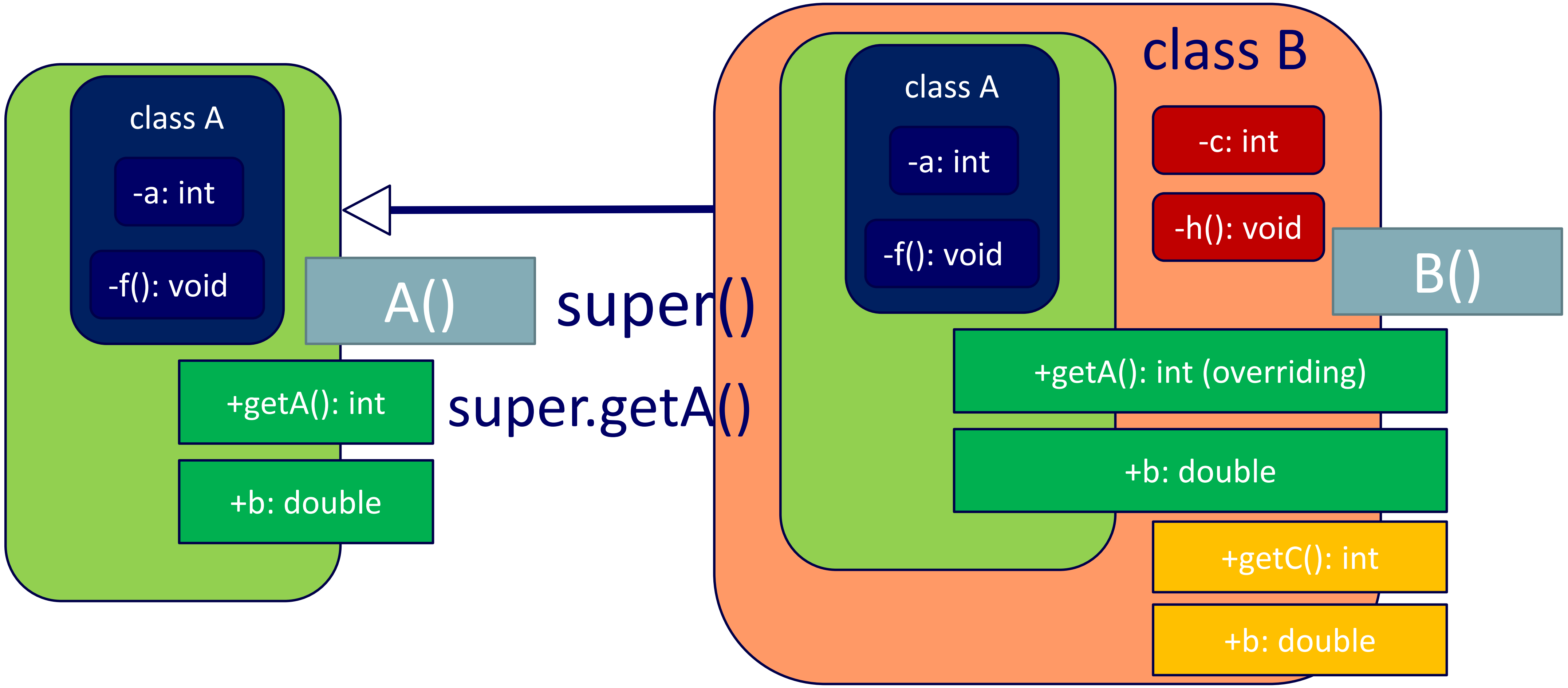
LECTURE 2

Defining a Subclass

A subclass inherits from a superclass. You can also:

- Add new properties
- Add new methods
- Derived new properties from base class
- Override the methods of the superclass

super keyword



super keyword

LECTURE 3

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:

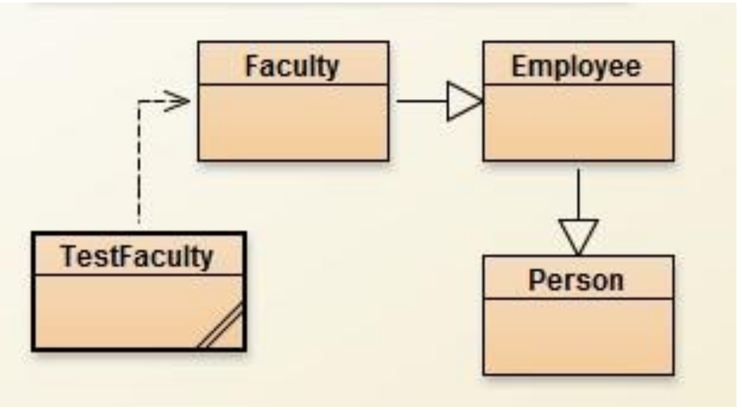
- To call a superclass constructor
- To call a superclass method (access public methods, protected methods, default methods same package only, no private methods)

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

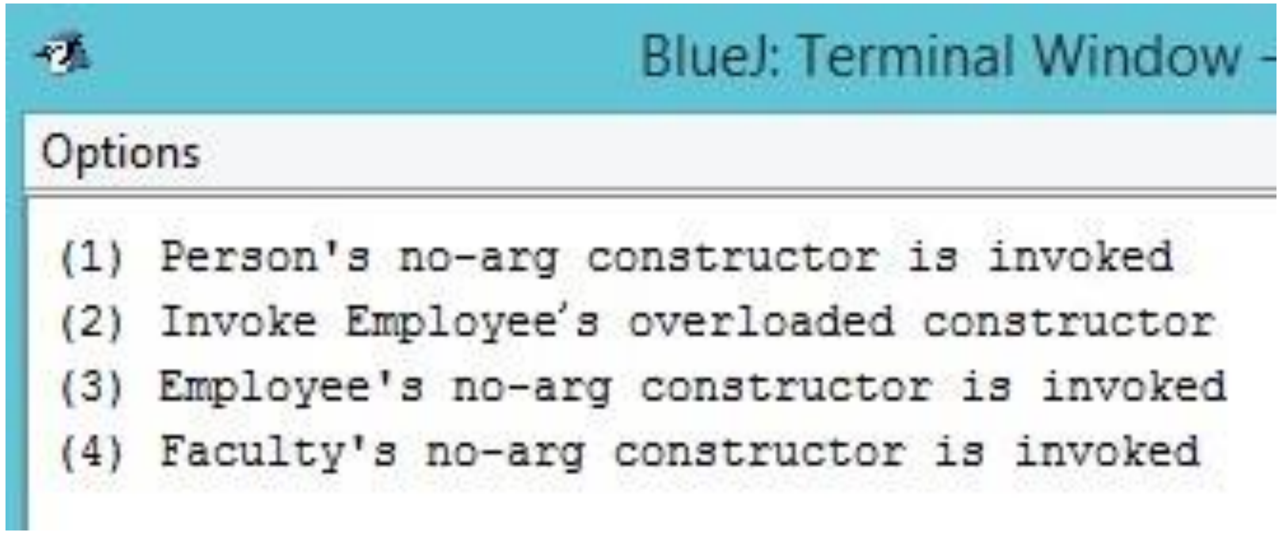
```
public class Faculty extends Employee {
  public static void main(String[] args) {
    new Faculty();
  }

  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



```
(1) Person's no-arg constructor is invoked
(2) Invoke Employee’s overloaded constructor
(3) Employee's no-arg constructor is invoked
(4) Faculty's no-arg constructor is invoked
```

Results:

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");
  }
}
```

Nothing will be called

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);
}
```

Example of Inheritance

LECTURE 4

Demonstration Program

BUILDING A NEW SUBCLASS – RECTANGLE, SQUARE, PRISM, OVAL, CIRCLE SPHERE

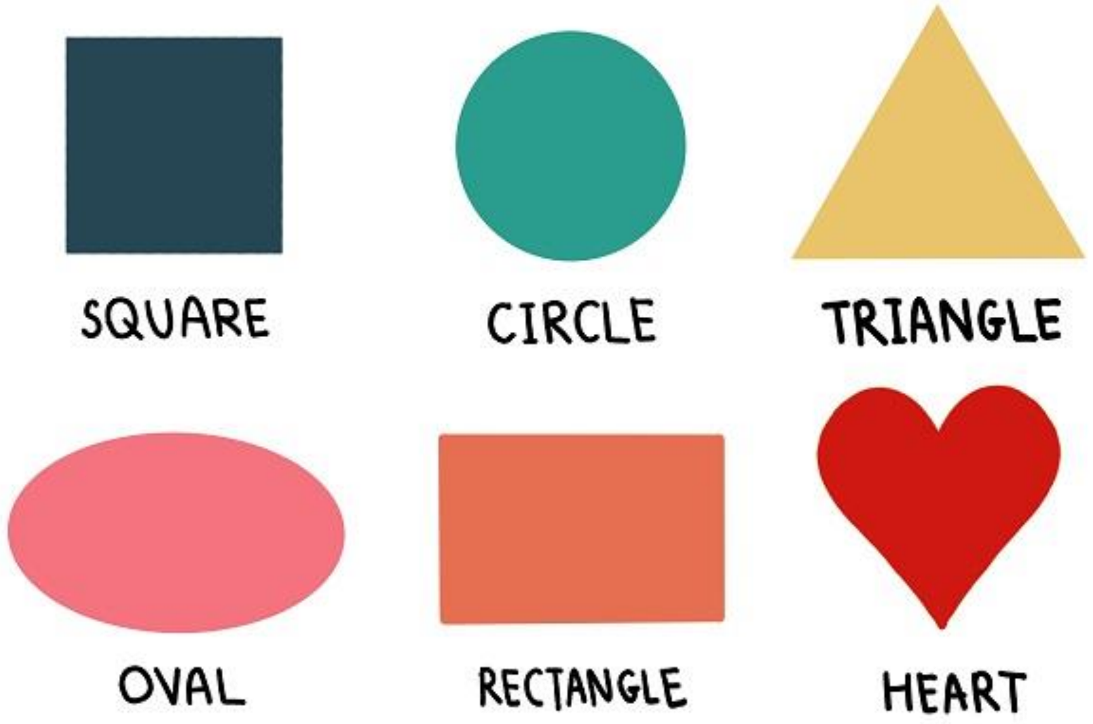
Multiple-Inheritance

LECTURE 5

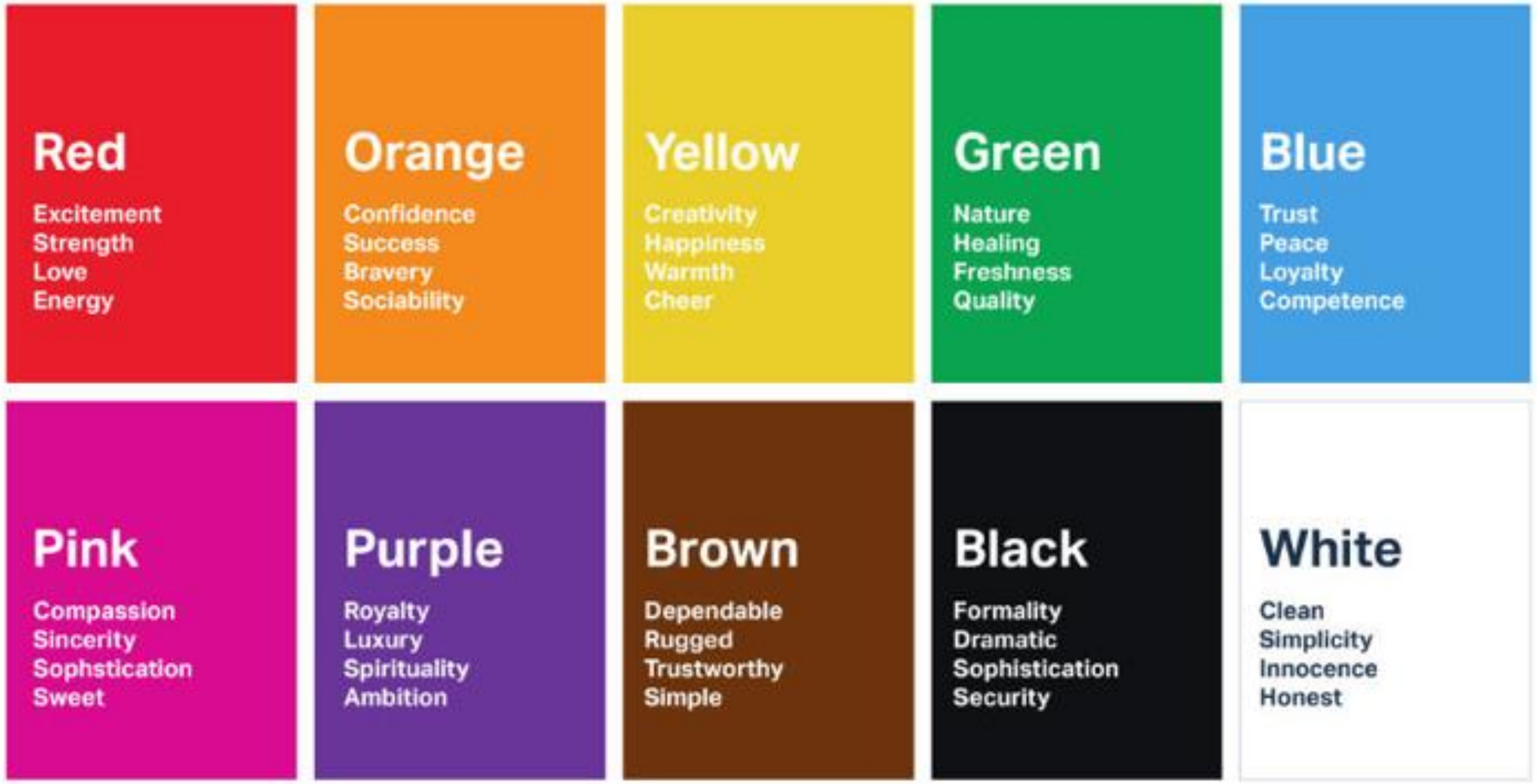
Multiple Inheritance

- **Multiple inheritance** means a class extends multiple parent classes.
- Java does not support multiple inheritance, but other languages do.
- To solve this problem, you have use interface (Java allows multiple-implementations of interfaces), or hierarchical inheritance to realize multiple inheritance.

Shape



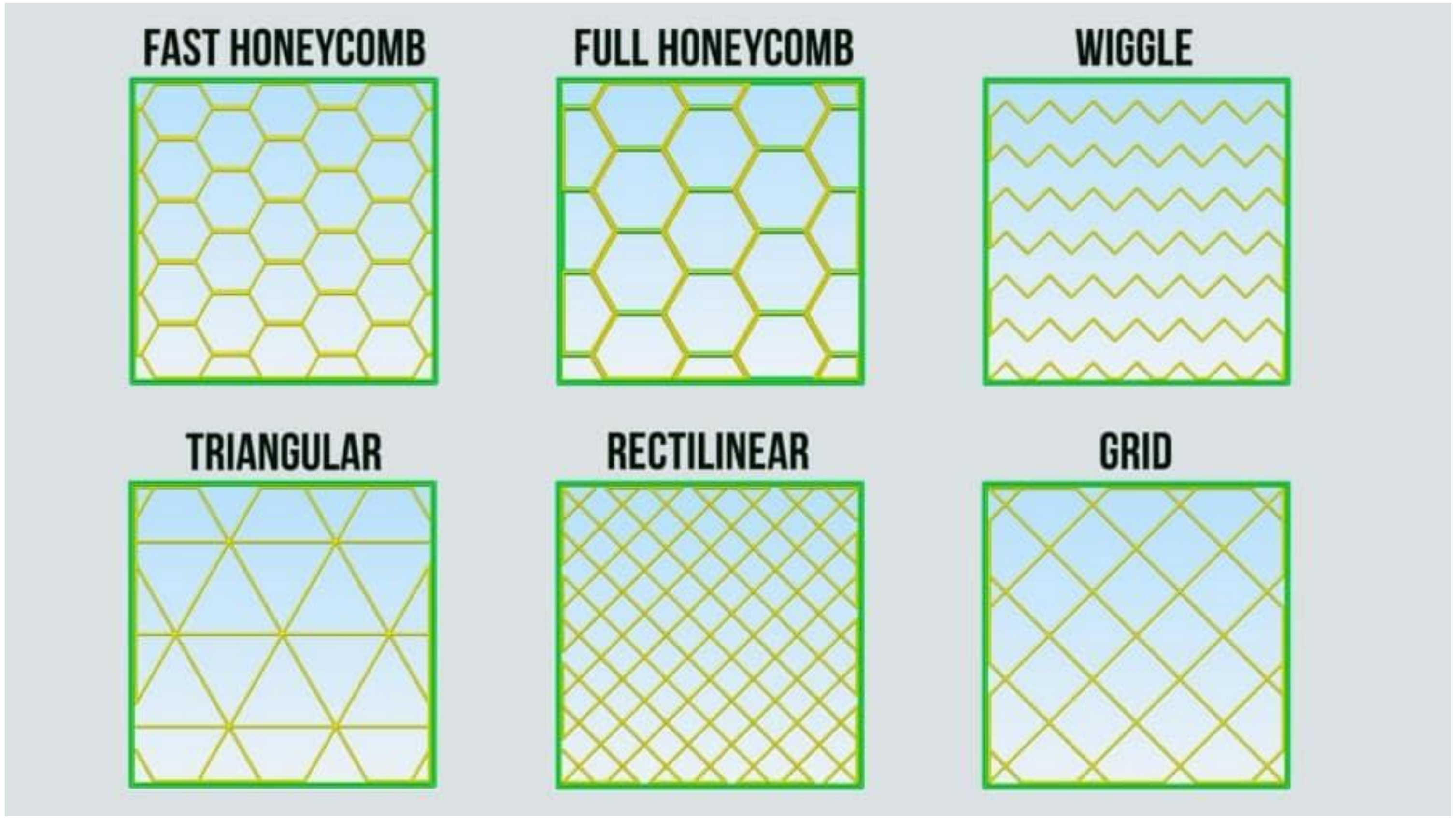
Paint Colors



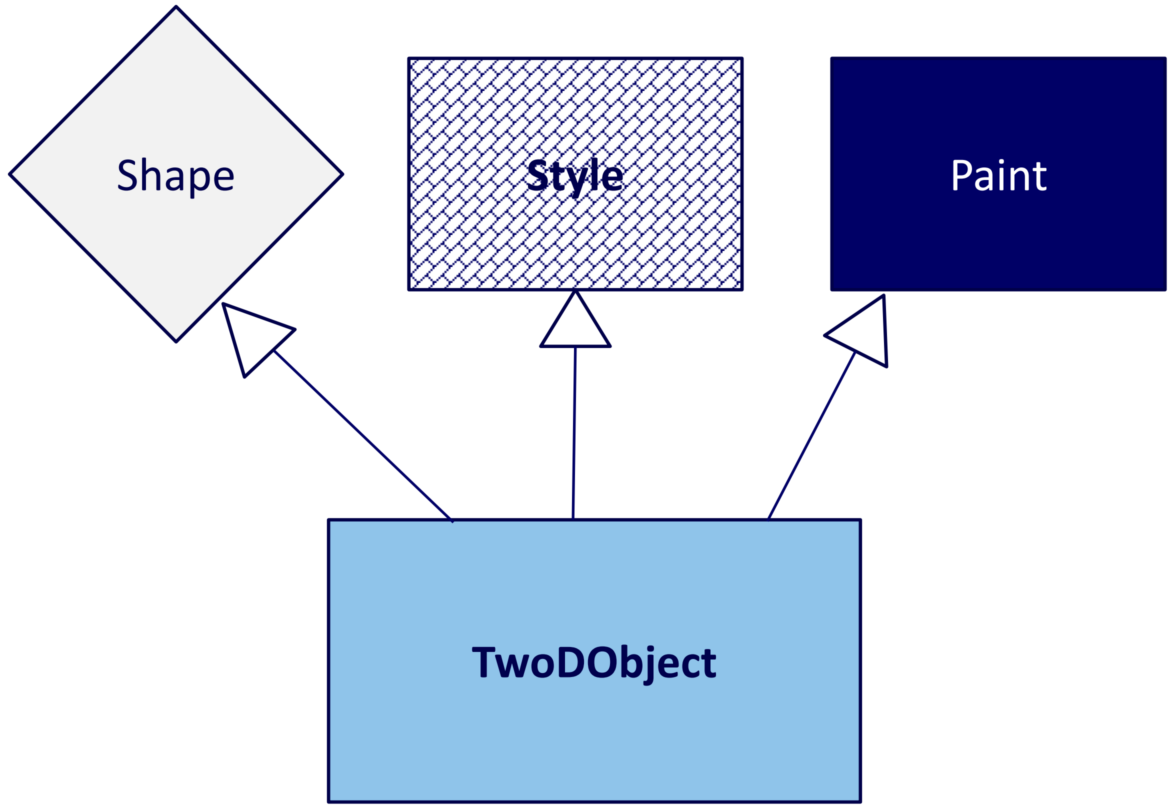
Red	Orange	Yellow	Green	Blue
Excitement	Confidence	Creativity	Nature	Trust
Strength	Success	Happiness	Healing	Peace
Love	Bravery	Warmth	Freshness	Loyalty
Energy	Sociability	Cheer	Quality	Competence

Pink	Purple	Brown	Black	White
Compassion	Royalty	Dependable	Formality	Clean
Sincerity	Luxury	Rugged	Dramatic	Simplicity
Sophstication	Spirituality	Trustworthy	Sophistication	Innocence
Sweet	Ambition	Simple	Security	Honest

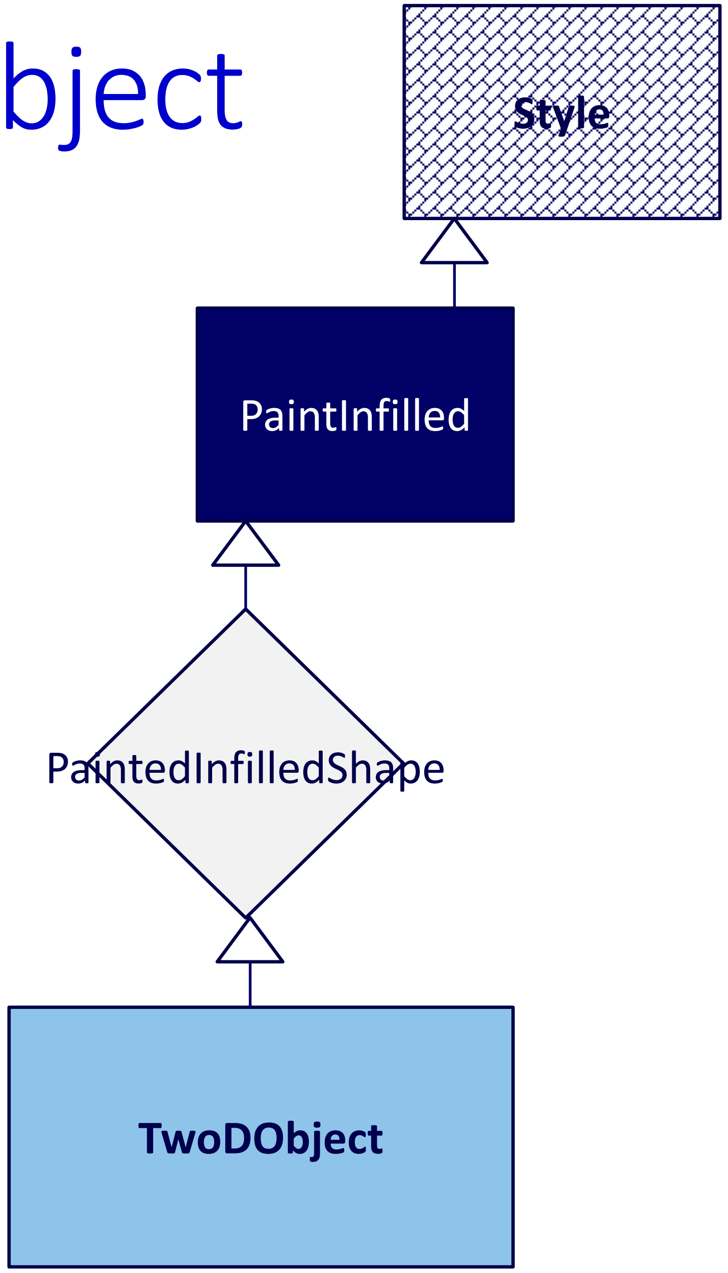
Infill Styles

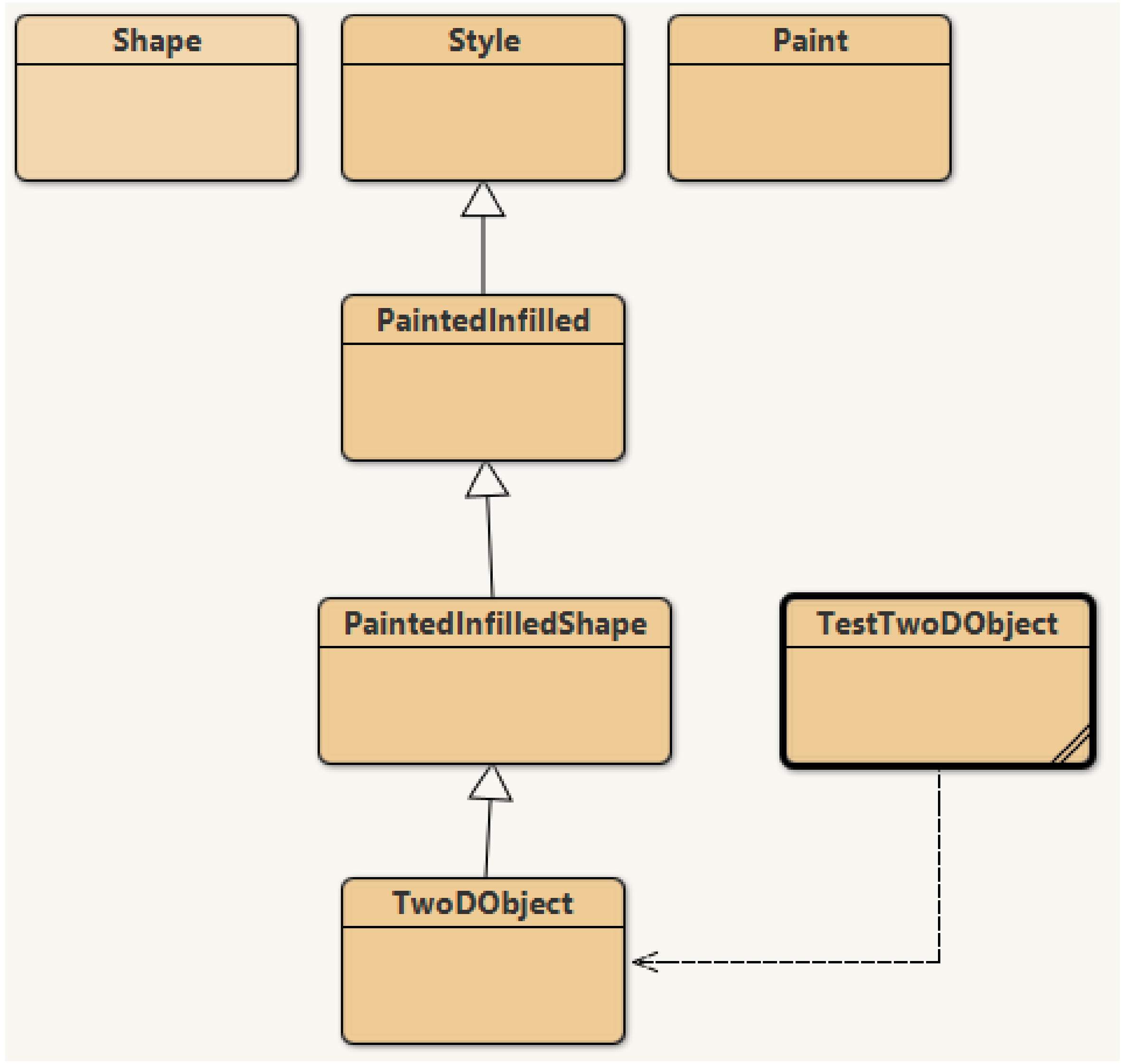


Geometric Object



Geometric Object




Shape
Style
Paint
PaintedInfilled
PaintedInfilledShape
TestTwoDObject
TwoDObject

Demonstration Program

SHAPE, STYLE, PAINT, PAINTEDINFILLED, PAINTEDINFILLEDSHAPE, TWODOBJECT, TESTTWODOBJECT

Reference and Object

LECTURE 6

Reference and Object

The **reference variable** can be reassigned to other objects provided that it is not declared final. The type of the reference variable would determine the methods that it can invoke on the object.

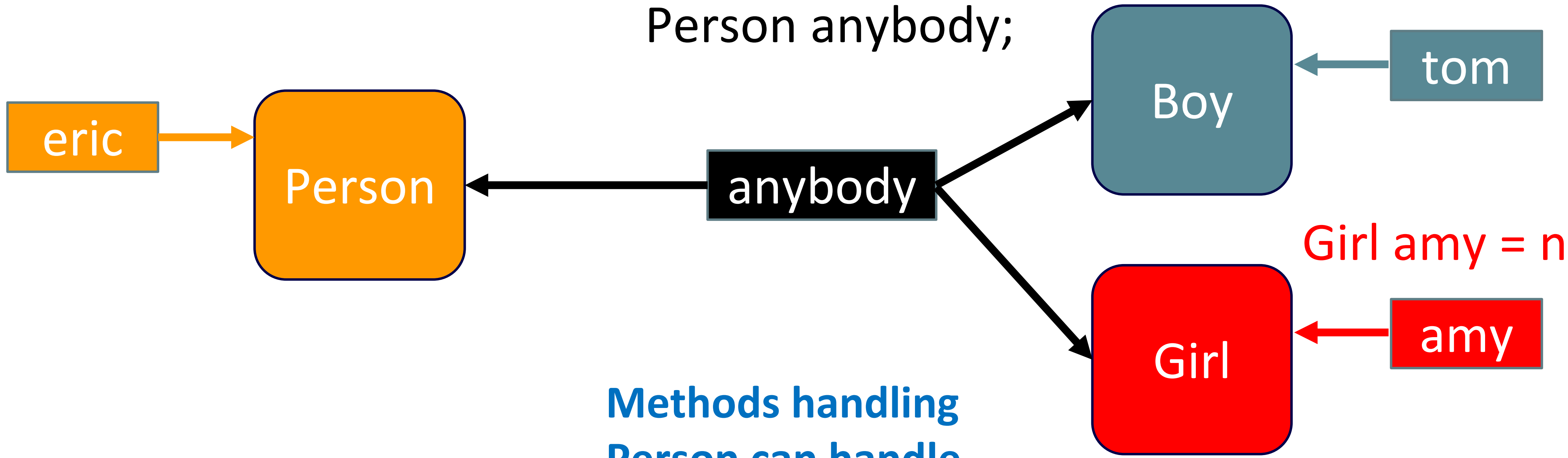
A reference variable can refer to any object of its declared type or any subtype of its declared type. A reference variable can be declared as a class or interface type.

```
Person eric = new Person();
```

```
Boy tom = new Boy();
```

```
Person anybody;
```

```
Girl amy = new Girl();
```

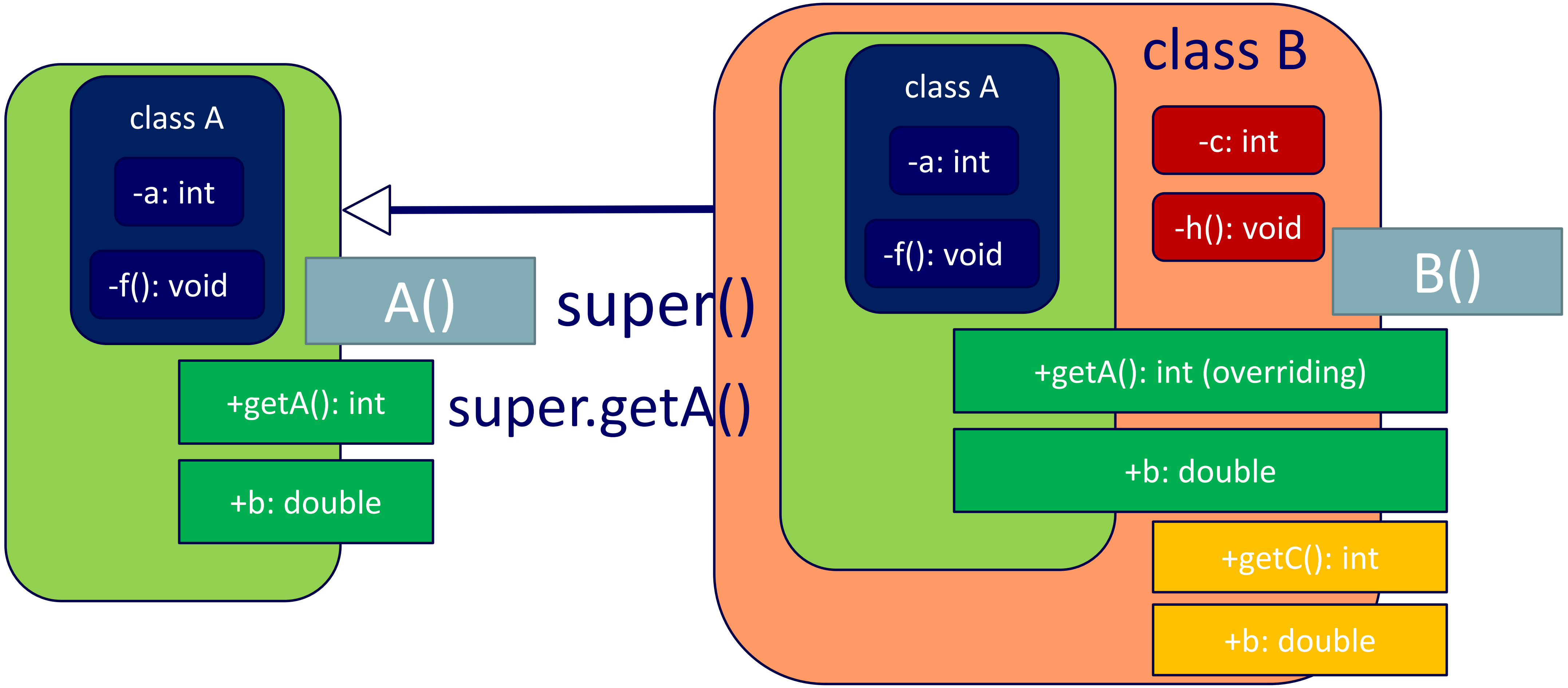


Methods handling Person can handle Boys and Girls.

Parent Class

Children Class

Reference


class A
-a: int
-f(): void
+getA(): int
+b: double
A()
super()
super.getA()
class B
class A
-a: int
-f(): void
-c: int
-h(): void
B()
+getA(): int (overriding)
+b: double
+getC(): int
+b: double