CSE 8B Today

Inheritance!

Person

Student

CSEMajor

JPannel

GraphicLetter

YOUR Graphic Letter

Inheritance:

Object

Dimension

Font

FontMetrics

Color

Graphics

Component

Container

LayoutManager

Panel

Applet

JApplet

Window

Frame

JFrame

Dialog

JDialog

JComponent

Swing GUI components such as JButton, JLabel, JTextField, JPanel, etc.

Swing Components in the javax.swing package

Lightweight

Heavyweight

Classes in the java.awt package
1. What kinds of figures can the Graphics class draw?

A. People, animals, vehicles.
B. Graphs, plots, histograms.
C. Buttons, frames, labels.
D. Shapes, images, strings.
2. The Polygon class has three public data fields (instance variables) called npoints, xpoints, and ypoints. This is a bad design. Why?

A. A Polygon class does not need to have an npoints data field.

B. Data fields in a class should be private, not public.

C. The Polygon class doesn’t actually use these data fields.

D. A class shouldn’t have data fields at all.
3. Where is the origin located for each component in the Java coordinate system (not the standard coordinate system)?

A. Lower right corner
B. Lower left corner
C. Upper left corner
D. Upper right corner
What will print in the main code here?

A. “Now offline: Sally”
B. “Now online: Sally”
C. “Now offline: Person@15c61fb5”
D. “Now online: Person@15c61fb5”
class Person
{
    protected String name; // data member - protected

    public Person(String name) { this.name = name; }
    public boolean isAsleep(int hr) { return 22 < hr || 7 > hr; }
    public String toString() { return name; }

    public void status(int hr)
    {
        if (this.isAsleep(hr))
            System.out.println("Now offline: " + this);
        else
            System.out.println("Now online: " + this);
    }
}

class Student extends Person
{
    protected int units; // additional data member

    public Student(String name, int units)
    {
        this.name = name;
        this.units = units;
    }

    public boolean isAsleep(int hr) // override
    { return 2 < hr && 8 > hr; }

    public String toString()
    {
        String result = super.toString();
        return result + " units: " + units;
    }
}

In main:
Student s;
s = new Student("Sally", 16);
s.status(1);
Inheritance Details

The constructor of the superclass will automatically be called by the subclass's constructor. If the superclass does not have a default constructor, you must explicitly invoke the super-class's constructor.

```java
class Person
{
    protected String name;  // data member - protected

    public Person( String name ) { this.name = name; }
    public boolean isAsleep( int hr ) { return 22 < hr || 7 > hr; }
    public String toString() { return name; }

    public void status( int hr )
    {
        if ( this.isAsleep( hr ) )
            System.out.println( "Now offline: " + this );
    }
}

class Student
extends Person
{
    protected int units;  // additional data member

    public Student( String name, int units )
    { super(name);
        this.units = units;
    }

    public boolean isAsleep( int hr ) // override
    { return 2 < hr && 8 > hr; }

    public String toString()
    {
        String result = super.toString();
        return result + " units: " + units;
    }
}

In main:
Student s;
s = new Student( "Sally", 16 );
s.status( 1 );
```
class Person
{
    protected String name;  // data member - protected

    public Person( String name ) { this.name = name; }
    public boolean isAsleep( int hr ) { return 22 < hr || 7 > hr; }
    public String toString() { return name; }

    public void status( int hr )
    {
        if ( this.isAsleep( hr ) )
            System.out.println( "Now offline: " + this );
        else
            System.out.println( "Now online: " + this );
    }
}

class Student extends Person
{
    protected int units; // additional data member

    public Student( String name, int units ) {
        super(name);
        this.units = units;
    }

    public boolean isAsleep( int hr ) // override
    { return 2 < hr && 8 > hr; }

    public String toString()
    {
        String result = super.toString();
        return result + " units: " + units;
    }
}

Inheritance Details

What will this code print?
A. Now online: Sally units: 16
B. Now offline: Sally units: 16
C. Now online: units: 16
D. Now online: units: 16
E. Nothing, there is an error

in main:
Student s;
s = new Student( "Sally", 16 );
s.status( 1 );
**Polymorphism**

```
class Person
{
    protected String name;  // data member - protected

    public Person( String name ) { this.name = name; }
    public boolean isAsleep( int hr ) { return 22 < hr || 7 > hr; }
    public String toString() { return name; }

    public void status( int hr )
    {
        if ( this.isAsleep( hr ) )
            System.out.println( "Now offline: " + this );
        else
            System.out.println( "Now online: " + this );
    }
}

class Student extends Person
{
    protected int units;  // additional data member

    public Student( String name, int units ) { super(name); this.units = units; }

    public void status( int hr ) { System.out.println( "I'm studying"); }
    public boolean isAsleep( int hr ) { return 2 < hr && 8 > hr; }

    public String toString()
    {
        String result = super.toString();
        return result + " units: " + units;
    }
}
```

```
Polymorphism

All students are people!

"We always start looking in the class for the object that we have"

What will this code print?
A. Now online: Sally units: 16
B. Now offline: Sally
C. This code has a compile error
D. This code has a run-time error
E. I don’t know

in main:
Person p;
p = new Student( "Sally", 16 );
p.status( 1 );
p.study(); < compile error
```

```
8

"Sally"
"I'm studying"

"Now online: Sally units: 16"
```
class Person
{
    protected String name;  // data member - protected

    public Person( String name ) { this.name = name; }
    public boolean isAsleep( int hr ) { return 22 < hr || 7 > hr; }
    public String toString()   { return name; }

    public void status( int hr )
    {
        if (this.isAsleep( hr ))
            System.out.println( "Now offline: " + this);
        else
            System.out.println( "Now online: " + this);
    }
}

class Student
extends Person
{
    protected int units; // additional data member

    public Student( String name, int units ) {
        super(name);
        this.units = units;
    }

    public boolean isAsleep( int hr ) // override
    { return 2 < hr && 8 > hr; }

    public String toString()
    {
        String result = super.toString();
        return result + " units: " + units;
    }
}

All Students are People. So it is OK to store a Student in a variable that refers to a Person, because you will never try to do something that a Student cannot do. This makes the compiler happy.

AT RUNTIME, Java determines that the OBJECT is actually a person, so it invokes the STUDENT version of the isAsleep method (from the Person’s status method).

Polymorphism

```java
Person p;
p = new Student( "Sally", 16 );
p.status( 1 );
```
Polymorphism

Sometimes the exact type is known at compile-time:

"Ordinary" code -- exact type known at compile time:

```java
Student s = new Student( "Sally", 16 );
then we can use s.units or s.isAsleep ...
```
But sometimes it's not known until **run-time:**
- The compiler will assume the object is of the **declared** type.

The constructor still determines the **actual** type of the Object.
- At run-time, Java will use the **actual type's** latest (**most-derived**) methods.

This is legal too (and may be very useful!):

```java
Person p;
later on…
p = new Student( "Sally", 16 );
```