CSE 8B Today

How’s PSA4?
A. I’m done!
B. I’m working on it
C. I’ve read it, but haven’t started coding
D. I haven’t even read it

Are all students people? Pondering life’s deep questions

Get it done ASAP! It’s a very good practice for next week’s test

Any issue with your participation grades?
First days missing cause you registered late?

Follow up on Piazza:
“Gradesource/iClicker issues”

Hierarchy of Classes
Emphasis on PSA3 and PSA4 topics
Scope, Objects, references and memory models
We can have

- static methods and variables
  (“belong” to the Class, so we **don’t need** an object to access them)

- Instance (NON-static) methods and variables
  (“belong” to an Instance, so **we need** an object to access them)
Static methods can be called directly on the class name (can also be called on an object of the right class, but it’s strongly recommended NOT TO DO IT)

WE WILL NOT BE CREATING A LOT OF “STATIC” ELEMENTS, so don’t worry much about them. THE MOST IMPORTANT THING FOR YOU TO REMEMBER FOR NOW IS THAT THEY DON’T NEED AN OBJECT TO CALL THEM to ACTUALLY RUN, so we will call them from the Class Name.

- ✓ Access instance variable
- ✓ Access static variable
- ✓ Call instance method
- ✓ Call static method

- X Access instance variable
- ✓ Access static variable
- X Call instance method
- ✓ Call static method
Beyond copy-and-paste

Java is an object-oriented programming language:

- **Classes**: user-defined datatypes
- **Objects**: variables of those types

The primary goal of OOP is to **create a good abstraction**

- one that models relationships accurately
- without forcing the user to keep track of more than necessary

There are **two** relationship types that Java can model . . .
Object-oriented programming

There are two relationship types that Java can model...

**Two approaches to re-use code in Java:**

**Object reuse** (what we have been doing ...)

```java
public class Make8BFrame {
    public static void main(String[] args) {
        JFrame myFrame = new JFrame("This is my window");
        myFrame.setSize(300, 400);
        myFrame.add(new JLabel("CSE 8B"));
        myFrame.setVisible(true);
    }
}
```

**Object specialization/extension** (what we are learning now ...)

```java
public class CSE8BFrame extends JFrame {
    public CSE8BFrame() {
        super("This is my window");
        this.setSize(300, 400);
        this.add(new JLabel("CSE 8B"));
    }
    public static void main(String[] args) {
        CSE8BFrame myFrame = new CSE8BFrame();
        myFrame.setVisible(true);
    }
}
```
Object-oriented programming

Two approaches to re-use code in Java:

Object reuse (what we have been doing ...)

```java
public class Make8BFrame {
    public static void main(String[] args) {
        JFrame myFrame = new JFrame("This is my window");
        myFrame.setSize(300, 400);
        myFrame.add(new JLabel("CSE 8B"));
        myFrame.setVisible(true);
    }
}
```

Why don’t we need to create a new JFrame object here?

A. We do. This code will cause an error.
B. We are not creating a window at all, so we don’t need a JFrame
C. The CSE8BFrame constructor does create a new JFrame object

Object specialization/extension (what we are learning now ...)

```java
public class CSE8BFrame extends JFrame {
    public CSE8BFrame() {
        super("This is my window");
        this.setSize(300, 400);
        this.add(new JLabel("CSE 8B"));
    }
    public static void main(String[] args) {
        CSE8BFrame myFrame = new CSE8BFrame();
        myFrame.setVisible(true);
    }
}
```
Inheritance Hierarchy

Any further?
Inheritance Hierarchy

```java
String name;

boolean isAsleep(int hr) {
    return hr > 22 || hr < 7;
}
```
Inheritance Hierarchy

Base Class
- Person

Derived Class
- Student

Very Derived Class
- CSEMajor

Data
- String name;
- int units;

Methods
- boolean isAsleep(int hr)
  {
    return hr > 22 || hr < 7;
  }

- boolean isAsleep(int hr)
  {
    return hr > 2 && hr < 8;
  }

  overriding the previous method
Inheritance Hierarchy

Person

Student

CSEMajor

Base Class

Derived Class

Very Derived Class

Data

String name;

int units;

boolean isTutor;

Methods

boolean isAsleep(int hr)
{
    return hr > 22 || hr < 7;
}

boolean isAsleep(int hr)
{
    return hr > 2 && hr < 8;
}
Think of a variable with a reference like a leash. Java thinks “On the other end of this leash, there has to be a Person”
Think of a variable with a reference like a leash. Java thinks “On the other end of this leash, there has to be a Person”

```java
Person p = new String( "Sally" );
```

NO! ERROR!
Person p = new Student( "Sally", 16 );

Will the line above cause an error?
A. Yes
B. No
Student students = new Person( "Sally");

Will the line above cause an error?
A. Yes
B. No
Variable types vs. object types

String s = new String( "Hello" );
Pixel p = new String( "This is a Pixel" );

Declared type

Object type

Must *(appear to)* match at compile time

Is there an error in the code above?
A. No, it’s fine
B. Yes, the first line will cause an error
C. Yes, the second line will cause an error
D. Yes, both lines would cause an error
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 );
  }
}

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”

Protected?
Visible for the Class and “derived” classes
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. Other (or error)
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;
    }
}

In main:
Person p;
p = new Student("Sally", 16);
p.status( 1 );

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
Variable types vs. object types

<table>
<thead>
<tr>
<th>Declared type (variable, reference) type</th>
<th>Actual type of the Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>Person p = new Student( “Sally”, 16 );</td>
<td></td>
</tr>
<tr>
<td>Student s = new Student( “Sally”, 16 );</td>
<td></td>
</tr>
<tr>
<td>p.status(1); // These two lines do</td>
<td></td>
</tr>
<tr>
<td>s.status(1); // EXACTLY the same thing</td>
<td></td>
</tr>
</tbody>
</table>

- **Declared type** and object type must (appear to) match at compile time
- At *compile time* Java makes sure that the methods and fields exist by using the variable (reference) type
- At *runtime*, Java uses the **ACTUAL type** of the object to determine which method to call (the variable type no longer matters)
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;
    }
}

Student s;
s = new Person( "Sally" );
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 )  {
        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;
    }
}

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 );

This code has an error. What is it?

A. The Person class has no empty constructor, which will cause an error in the Student constructor
B. The member variable name cannot be accessed in the constructor for Student
C. super.toString() in the Student’s toString method is undefined
D. The student class has no status method defined in it
Polymorphism

Sometimes an exact type is not known until \textit{run-time}:
- The compiler will assume the object is of the \textit{declared} type.

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

A: \texttt{Person }\texttt{p = new Student("Sally", 16 );}
B: \texttt{System.out.println("p's name is "} + \texttt{p.name);}
C: \texttt{System.out.println( p.isAsleep( 24 ));}
D: \texttt{p.status( 24 );}
E: \texttt{System.out.println("P is taking " + p.units);}

One of these lines of code will cause an error. Which one? (And how do you fix it)