Welcome to Connect Four!

DEBUG!
DEBUG!

| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
-------------
0 1 2 3 4 5

Debugging output is great, but **delete your debugging output before submitting**.

PSA 5: Show off your artistic side
(It’s easier, as promised BUT DON’T WAIT TO START!)

Looking for a partner? Try Piazza!

**Search for Teammates!**
- 7 Open Teammate Searches

How is PSA4 going?
A. Haven’t started
B. Got some of it done
C. Finished! (or mostly done)
D. And on to the extra credit!!
1. How many superclasses can a Java class inherit from using the extends keyword?

A. None
B. Only one
C. Only two
D. Many

B. Only one
2. You create a subclass method that has the same name, but different signature from that of the superclass. Is this overloading or overriding? Why/Why not?

A. Overloading, because it has a different signature from its superclass counterpart.

B. Overloading, because it has the same name as its superclass counterpart.

C. Overriding, because it has a different signature from its superclass counterpart.

D. Overriding, because it has the same name as its superclass counterpart.
3. The class Altima is a subclass of Nissan, and the class Nissan is a subclass of Car. True or false: Every Nissan object is also an Altima object. Why?

A. True, because the subclass Altima might have properties and methods that the Nissan object does not.

B. False, because the subclass Altima might have properties and methods that the Nissan object does not.
Scanner input = new Scanner(System.in);
System.out.println( this ); // print the board
System.out.println( "Player " + currPlayer + "," + "s turn." );
int row = -1;
int numSticks = -1;

while ( ___________________________________________ ) {
    System.out.print( "Which row? " );
    row = input.nextInt();
    System.out.print( "How many sticks? " );
    int numSticks = input.nextInt();
}

Which line correctly completes the code above so that the game will re-prompt
the user until they enter a legal row and number of sticks? isEmpty tests to see if the
whole board is empty. allowsMove tests to see if a given number of sticks can be taken
from a given row. ! means “not”
A. !this.isEmpty()
B. !this.allowsMove( row, numSticks )
C. numSticks <= 0
D. row <= 0
E. numSticks <= 0 || row <= 0
public class C4Board {
    private int height;
    private int width;
    private char[][] board;
    public C4Board( int width, int height )
    {
        width = width;
        height = height;
        // Set up the board here (code not shown)
    }

    public static void main( String[] args )
    {
        C4Board game1 = new C4Board(4, 6);
        System.out.println("# of Rows: "+ game1.height +
        "; # of Cols: " + game1.width );
    }
}

What does this print?
A. # of Rows: 0; # of Cols 0
B. # of Rows: 4; # of Cols 6
C. # of Rows: 6; # of Cols 4
D. Nothing, there is a compile error
E. Nothing, there is a NullPointerException exception
public class C4Board {
    private int height;
    private int width;
    public C4Board( int width, int height ) {
        width = width;  
        height = height;  
    }
    public static void main( String[] args ) {
        C4Board game1 = new C4Board(4, 6);
        System.out.println( "# of Rows: " + game1.height + "; # of Cols: " + game1.width );
    }
}
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:
Two approaches to reusing code in Java:

**Object reuse**

```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**

```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

Base Class: Person

Derived Class (subclass): Student

Very Derived Class (subclass): CSEMajor

Data:

- String name;

Methods:

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

Person

Base Class

Student

Derived Class

CSEMajor

Very Derived Class

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

- **Base Class**: Person
  - **Data**: String name;
  - **Methods**: boolean isAsleep(int hr)
    ```java
    {
        return hr > 22 || hr < 7;
    }
    ```

- **Derived Class**: Student
  - **Data**: int units;
  - **Methods**: boolean isAsleep(int hr)
    ```java
    {
        return hr > 2 && hr < 8;
    }
    ```

- **Very Derived Class**: CSEMajor
  - **Data**: boolean isTutor;
  - **Methods**: ??
Inheritance Hierarchy

Person

Student

CSEMajor

**Base Class**

**Data**

String name;

**Methods**

boolean isAsleep(int hr)
{
    return false;
}

**Derived Class**

int units;

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

**Very Derived Class**

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

boolean isTutor;

boolean isAsleep(int hr)
{
    return false;
}
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"

in main:
Person p;
p = new Person( "Sally" );
p.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;
    }
}

This code has an error. What is it?
A. The Person class has no default 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

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

Student s;
s = new Student( "Sally", 16 );
s.status( 1 );

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.
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 );
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).
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: 0
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:
Student s;
s = new Person( "Sally" );
s.status( 1 );
Inheritance Details

All People are NOT Students. If you store a Person in a variable that is supposed to reference a Student, the compiler gets nervous (and gives you an error) because you might try to make that object do something that it cannot do. This is true EVEN IF you never actually ask it to do something it cannot do (as in this example).

in main:
Student s;
s = new Person( "Sally" );
s.status( 1 );