We’re in the middle of the quarter! Let’s review what we (YOU but ME AS WELL!!) can improve.

What you like/dislike in/from....

• PSAs
• Group discussions/clicker questions (including the fact that it’s required)
• Lectures
• Lecture slides
• Starting PSAs early
• Tutors
Polymorphism

But sometimes an exact type is 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.

A: Person p = new Student( "Sally", 16 );
B: System.out.println("p's name is " + p.name);
C: System.out.println( p.isAsleep( 24 ) );
D: p.status( 24 );
E: 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)
Casting!

But sometimes an exact type is 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.

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

Casting changes the type of the reference to the object to make the compiler believe us that the object can really do what we ask it. Casting can be dangerous, so be sure you know what you’re doing!
Casting changes the type of the reference (NOT THE OBJECT)

Person $p = $new Student( "Sally");
Casting changes the type of the reference (NOT THE OBJECT)

"Trust me, this object is actually a Person"

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

The cast makes the object *look* like a plain Person at compile time. But the object is actually still a Student.
Casting changes the type of the reference (NOT THE OBJECT)

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

The cast makes the object *look* like a plain Person at compile time. But the object is actually still a Student.

At runtime, Java “applies the cast” to the object, which does not change the type of the object, but will result in an error if the types don’t match.
Casting changes the type of the reference (NOT THE OBJECT)

```java
Person p = new Person( "Sally" );
Student s = p;
```

Will this code cause an error?
A. Yes, at compile time       B. Yes, at run time       C. No
Casting changes the type of the reference
(NOT THE OBJECT)

`Person p = new Person( "Sally" );
Student s = (Student)p;`

Will this code cause an error?
A. Yes, at compile time  B. Yes, at run time  C. No
ArrayLists
public void setUniqueAndCounts( String[] words )
{
    int index = 0;
    int uniqueIndex = 0;
    int numUniqueWords = countUnique( words );
    uniqueWords = new String[numUniqueWords];
    counts = new int[numUniqueWords];
    while ( index < words.length )
    {
        int count = getCount( words, index );
        uniqueWords[uniqueIndex] = words[index];
        counts[uniqueIndex] = count;
        index += count;
        uniqueIndex++;
    }
}

Why did we have to count the number of unique words before filling in the uniqueWords and counts arrays in PSA3?
A. We didn’t. This is just better style.
B. Arrays are fixed size, so we had to know how big to make them
C. We needed the number of unique words to control the while loop
If we would use **ArrayLists** ...

```java
public ArrayList counts;
public ArrayList uniqueWords;

public void setUniqueAndCounts( String[] words )
{
    counts = new ArrayList();
    uniqueWords = new ArrayList();
    int index = 0;
    int uniqueIndex = 0;
    while ( index < words.length )
    {
        int count = getCount( words, index );
        uniqueWords.add(words[index]);
        counts.add( new Integer( count ) );
        index += count;
    }
}
```

So we need a *Wrapper class* (An object that holds a primitive value)
Java will do this automatically if you add an int to an ArrayList

ArrayLists **only hold Objects**, NOT PRIMITIVES
Another Use for Casting: ArrayLists

```java
public void displayWords()
{
    JFrame wordFrame = new JFrame("Word cloud");
    wordFrame.setLayout(new FlowLayout(FlowLayout.LEFT));

    for ( int i = 0; i < uniqueWords.size(); i++ ){
        int fontScale = 2 * counts.get( i );

        // More code not shown
    }
}
```

The last line above has an error. Why?

A. get is not the right method to get an element from an ArrayList
B. counts.get(i) will return an Integer, which cannot be multiplied by 2
C. counts.get(i) returns a reference to an Integer, which when multiplied by 2
   gives another Integer, and not an int
D. counts.get(i) will return a reference to an Object, which cannot be multiplied
   by 2
Abstract class?
Abstract class example...

```java
public class GraphicLetter extends JPanel {
    public char whichChar() {
        return '\u0000';
    }

    public GraphicLetter makeCopy() {
        return null;
    }
}
```

These methods are “dummy” methods

```java
public abstract class GraphicLetter extends JPanel {
    public abstract char whichChar();
    public abstract GraphicLetter makeCopy();
}
```

In an abstract class we need not even define “dummy” methods.
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;
    }
}
class CSEMajor extends Student
{
    protected boolean isTutor;

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

    public CSEMajor(String name, int units, boolean isTutor)
    {
    }

    public String toString()
    {
    }

    public static void main(String[] args)
    {
        Student s = new Student("Sally", 16);
        s.status(7); // status at 7 am

        Person p = new CSEMajor("Susan", 18, true);
        p.status(3); // status at 3 am

        Student s = p;
        s.status(3); // status at 3 am
    }
}
Towards the Connect 4 GUI... DEMO!

NEXT WEEK! PSA6 Released on Thursday We’ll talk about it on Tuesday