Hey, that's 26 points in Scrabble!

“Strings” and ‘c’ ‘h’ ‘a’ ‘r’ ‘s’ in Java

PSA2: Implementing your very own Secret Decoder Ring in Java...

... and a little Scrabble to warm up

Will be posted by this evening.  START EARLY!
1. Is a Java String an Object or a primitive type? Is a Java char an Object or a primitive type?

A. String is an Object, char is an Object

B. String is an Object, char is a primitive

C. String is a primitive, char is an Object

D. String is a primitive, char is a primitive
2. What Java statement will test whether two variables of type String, s1 and s2, store the same value?

A. `s1.equals(s2)`
B. `s1 == s2`
C. `s1.sameAs(s2)`
D. `s1 && s2`

```java
String s1 = new String("Hello");
String s2 = new String("Hello");
s1 == s2; // false
char c1 = 'A';
c2 = 'A';
```
2. You have a variable named `myString` that stores the String "CSE8B Rocks!". What Java statement will return the substring "Rocks" from `myString`?

A. `myString.substring(0)`

B. `myString.substring(6)`

C. `myString.substring(6, 11)`

D. `myString.concat( "Rocks" )`
4. You have created a variable named `myCharacter` as follows:

```java
Character myCharacter = new Character('M');
```

The value of `myCharacter.compareTo('N')` is `-1`. Explain why this is true.

A. N’s Unicode (ASCII) code is one greater than M’s

B. -1 is returned for any letter that comes after M in the alphabet

C. -1 is an error code returned by this method
public class StringPlay {
    public boolean hasLetter( String word, char letter ) {
        for ( int i = 0; i < word.length(); i++ ) {
            if ( word.charAt( i ) == letter ) {
                return true;
                System.out.println( "Found it!" );
            }
        }
        return false;
    }
}

StringPlay s = new StringPlay();
System.out.println( s.hasLetter( "Sleep", 'S' ) );
public class StringPlay {
    public boolean hasLetter(String word, char letter) {
        for (int i = 0; i < word.length(); i++) {
            if (word.charAt(i) == letter) {
                return true;
            }
        }
        return false;
    }
}

StringPlay s = new StringPlay();
System.out.println(s.hasLetter("Sleep", 's'));

What will be printed?
A. Nothing: there is a compile error
B. "false"
C. "true"
D. I don’t know

How could we ignore case?
notice the case (lower)
chars Big and Small

```java
public class StringPlay {
    public boolean hasLetter(String word, char letter) {
        for (int i = 0; i < word.length(); i++) {
            if (word.charAt(i) == letter) {
                return true;
            }
        }
        return false;
    }
}

StringPlay s = new StringPlay();
System.out.println(s.hasLetter(“Sleep”, ‘s’));
```

If we want this to return true, how could we ignore case?
The **Character** class (very useful—check your book)

```java
public class StringPlay {
    public boolean hasLetter( String word, char letter ) {
        letter = Character.toLowerCase( letter );
        for ( int i = 0; i < word.length(); i++ ) {
            char current = Character.toLowerCase( word.charAt( i ) );
            if ( current == letter ) {
                return true;
            }
        }
        return false;
    }
}
```
public class StringPlay {
    public boolean hasLetter( String word, char letter ) {
        letter = Character.toLowerCase( letter );
        for ( int i = 0; i < word.length(); i++ ) {
            char current = Character.toLowerCase( word.charAt( i ) );
            if ( current == letter ) {
                return true;
            }
        }
        return false;
    }
}

StringPlay s = new StringPlay();
String check = new String( "Sleep" );
s.hasLetter( check, 's' );
System.out.println( check );

What will be printed?
A. “Sleep”
B. “sleep”
C. “true”
D. “false”
E. I don’t know
Strings are immutable

```java
public class StringPlay {
    public boolean hasLetter(String word, char letter) {
        letter = Character.toLowerCase(letter);
        for (int i = 0; i < word.length(); i++) {
            char current = Character.toLowerCase(word.charAt(i));
            if (current == letter) {
                return true;
            }
        }
        return false;
    }
}

StringPlay s = new StringPlay();
String check = new String("Sleep");
s.hasLetter(check, 's');
System.out.println(check);
```

If you WANT to change a string, you must make a new string
Complete the following method to replace all instances of one character with another in a string (and the return the resulting string). Your method should be case sensitive.

```
e.g., replaceChar( "One for the money, YO!", 'O', 'i' ) →
        "ine for the money, Yi!"
```

```
public String replaceChar( String s, char gone, char here )
{
    char[] chars = s.toCharArray();
    for (int i = 0; i < chars.length; i++) {
        if (chars[i] == gone) {
            chars[i] = here;
        }
    }
    return String.valueOf( chars );
}
```

Extra: Can you replace both lower and upper case instances and preserve case?
Creating new Strings (worksheet)

Complete the following method to replace all instances of one character with another in a string (and the return the resulting string). Your method should be case sensitive.

e.g., replaceChar( “One for the money, YO!”, 'O', 'i' ) → “ine for the money, Yi!”

public String replaceChar( String s, char gone, char here )
{
    char[] chars = s.toCharArray();
    for ( char c : chars ) {
        if ( c == gone ) {
            c = here;
        }
    }
    return String.valueOf( chars );
}

Extra: Can you replace both lower and upper case instances and preserve case?
Chars and ints: deceptively similar

What is the value of each of the expressions below:

(1) > (int)'A'
(2) > 'A' + 2
(3) > (char)('A' + 2)

Remember, casting a value changes its type

A. error, error, error
B. 65, error, error
C. 65, ‘C’, ‘C’
D. 65, 67, ‘C’
E. I don’t know
Chars and ints: deceptively similar

What is the value of each of the expressions below:

(1) > (int)'A'
(2) > 'A' + 2
(3) > (char)('A' + 2)

but apparently types change even without a cast

Important: when you add an int to a char, it automatically becomes an int. To get a char you have to cast it back to a char.