Copy pgs 148 - 154 assign for reading

black java book lamp analogy - read again before teaching.
i love lamp video clip.

pg 150 - 151

type in Lamp.java and LampTest → to use during lecture

AP CS Unit 4: Classes and Objects Notes

An object is something that contains both data and methods. What data and what methods depend on its class. A class is generally used as a "blueprint" to create objects. In technical terms, a class models/defines the data and behavior associated with some entity.
is a blueprint

Example:

Server the class	**Client** the program to run a class
public class Square {	public class Runner {
private int side;	public static void main(String[] args) {
public Square(int x) { side = x; }	constructor
public int area() { int a = side*side; return a; }	accessor method
public void set(int x){ side = x; }	mutator method - an instance variable is changed.
}	} }

A class consists of:

Instance variables: for Square this is side only. notice it is private. we will always do this!

Constructors Square(int x) has parameter
Keep in mind that constructors can be numerous!
(may have different parameters as well)

idea of lamp vs a specific lamp.

An object is an instance of a class. A constructor is used to construct/instantiate an object.

Methods area() this method returns an integer value

Visibility Modifiers

public outside clients / programs can use these to perform a job on/with object

private only accessible within class itself accessor (returns a value) / mutator (changes state of object)

Methods have three parts: a type, visibility modifier, and parameters.
Here are the two methods of the Square class.

→ accessor, returns an int   → indicates no return value

public int area() {
    int a = side*side;
    return a;
}
	public void set(int x){ mutator method
	    side = x;
	}

A **return statement** causes the program to exit the method. Let's consider three examples:

void methods don't require return stmt. the compiler adds it! ☺

If we call method1 and n equals 3, what does the method return? does not compile. otherwise 12
 	
If we call method1 and n equals -22, what does the method return? d n c otherwise 6
	public double method1(int n) { 3
	    if (n < 0) {
	        return 6;
	    }
	    int b = 4*n;
	    return b;
	}

If we call method2 and the parameters are 3 and 5, what is displayed? hey
 	
Notice that this return statement (1) returns nothing and (2) causes the program to exit early.
	public void method2(int a, int b) { 3 5
	    int c = a + b;
	    if (c % 2 == 0) {
	        System.out.println("hey");
	        return;
	    }
	    System.out.println("joe");
	}

method3 causes this compiler error:
    unreachable statement
 	
What does this mean? c = c + 3 is never executed
	public int method3(int c) {
	    c = c + 2;
	    return c;
	    c = c + 3;
	}

$\frac{c}{8}$

In the context of a class, all variables fall into one of three categories:

- instance variables
- parameters
- local variables

Name any instance variables side
Name any parameters int x
Name any local variables int a in area( )
	public void method4(int h) {
	    int g = h + k;
	}

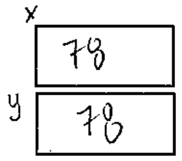
Only one line generates the compiler error:
    variable ____ might not have been initialized
 	
Which line is it? 4
	public void method5(int n) { // 1
	    int a; // 2
	    System.out.println(n); // 3
	    System.out.println(a); // 4
	}

modifier **Two Common Terms.** If a method changes the value of an instance variable, then it is called a mutator or instance method. If a method returns the value of an instance variable, then it is called an accessor or return method.

The Difference between Primitive and Object (aka Reference) Data Types

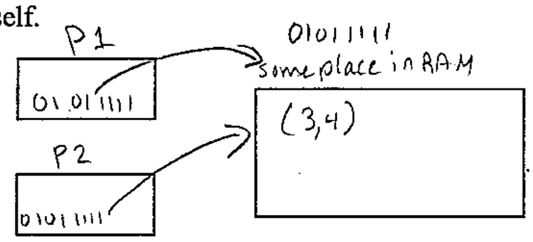
Types. First, remember the definition of a variable. A variable is a storage place. So, an obvious question is: what is stored in a particular variable? For primitive data types (e.g. int and double) the answer is easy. The variable stores the data.

```
int x = 78;
int y = x;
```



For object variables, the answer is more complicated. An object variable contains a reference to an object - not the object itself.

```
Point p1 = new Point( 3, 4 );
Point p2;
p2 = p1;
```



1. How many Point objects does this code create? 0	Point x1;
	Point x2;

2. How many Point objects does this code create? 1	Point x3 = new Point(17, -22);
	Point x4 = x3;

null is a special value that can assigned to any object variable. This indicates that the variable no longer contains a reference to an object. For example: doesn't point to any space in memory

```
Point p1 = null;
```

If an object is instantiated but at some later time no variable contains a reference to that object, the JVM will delete that object in a process called garbage collection. For example:

```
Point man = new Point( -1, 0 );
man = null;          // point to nothing
```

If you attempt to call a method but the object variable does not contain a reference to an object, then you will get this runtime error: null pointer error. For example:

```
Point pt = null;
double d = pt.distance( 0, 5.5 );        // run time error
```

Another Data Type. So far we have used two types of primitive variables: ints and doubles.

Another data type is the boolean data type. Variables of type boolean have a value of true or false. Wherever you can use a boolean expression, you may also use a boolean variable.

booleans can show "state" of an object better.

Example 1. Here is one way a boolean variable may be used. In this context it is called a conditional or loop control because it signals when to keep playing and when to stop.

```
boolean game_on = true;
while ( game_on ){
      int n = (int) (7*Math.random()) + 2;
      System.out.println( n );
      if ( n == 4 )
            game_on = false;
}
```

Example 2. boolean is a common parameter type and return type for methods.

have them enter & run this

```
import java.util.Scanner;

public class Runner {
    public static void main(String[] args) {
        Scanner in = new Scanner( System.in );
        System.out.print( "Enter a number: " );
        double n = in.nextDouble();

        Calculator c = new Calculator();
        c.set( true );   // turns calc on
        double x = c.squareIt( n );
        System.out.println( x );
    }
}
```

```
public class Calculator {
    private boolean on;

    public Calculator() {
        on = false;   <- when constructed calc is off
    }

    public double squareIt( double a ) {
        if ( on )
            return a*a;
        else
            return -1;
    }

    public void set( boolean b ) {
        on = b;
    }
}
```

The Not Operator. We have already covered the AND (&&) and OR (||) operators. A third logical operator is the NOT (!) operator. Here are some examples:

1. What is displayed? false	boolean boo = true;
	boo = !boo;
	System.out.println(boo);

2. Will the code execute if a = 4, b = 4, and c = 8? yes
 	
3. Name values that will cause the loop to terminate. c = 4 also
	while (!(a == b && b == c)){
	      // code
	}
	(a = 4, b = 4, c = 8)

explain expr false ! turns to true so while(true) runs!

a = b = c

primitives:
byte
short
int ← ☆
long
float
double ← ☆
char
boolean

☆ javasubset

char character not on java subset for AP CS

Char 16 bits (unsigned) 2^{16} possibilities

the variable is holding a reference to the object

The String Class. $\boxed{\text{A String object}}$ represents a sequence of one or more characters where a character could be a letter, digit, or punctuation mark. Each character in a string has a unique index starting at 0 ← very important

String s = "jump now"; // the *u* is at index 1, the *n* is at index 5

Commonly Used String Methods

Signature/Header	Example
int length()	int i = "mod 4/5".length(); System.out.println(i); prints 7
☆ if not found, returns a -1 ☆	

Signature/Header	Examples
int indexOf(String s) will find first occurrence of	String s1 = "bubbles"; int x = s1.indexOf("b"); 0 int y = s1.indexOf("bb"); 2 System.out.println(x + ", " + y); 2
int indexOf(String s, int i) ☆ in this string, starting at this index.	String s1 = "bubbles"; int x = s1.indexOf("b", 1); 2 int y = s1.indexOf("bb", 3); -1 System.out.println(x + ", " + y); 1
note: strings are immutable objects. notice below we create a new string to hold part of original string.	immutable - unable to change

Signature/Header	Examples

String substring(int start)	String s1 = "phone";
	String s2 = s1.substring(2); s2 = one
	System.out.println(s2);

String substring(int start, int end)
☆ up to but not including end ☆
	String s1 = "phone";
	String s2 = s1.substring(0, 2);
	System.out.println(s2); ph o

Comments	0 up to but not including 2, means positions 0 & 1 so 2 characters are printed.

Signature/Header	Example

boolean equals(Object obj)
comes from Object class, a method.
	String s1 = "a";
	String s2 = "A";
	if (s1.equals(s2))
	    System.out.println("ok");

== checks equality of primitive data types
if you use == w/ objects, you will merely be comparing the reference pointers. ie s1 == s2 will see if s1 and s2 point to the same space in memory! ("referential equality")
in string class this means s1.equals(s2) if the strings are exactly the same including case. mention .equalsIgnoreCase()

Signature/Header	Example

String toLowerCase()	String s1 = "4 SALE!";
	s1 = s1.toLowerCase();
	System.out.println(s1);

The above example prints 4 sale! This method did not change s1. It returned a reference to a new string that was assigned to s1. bc strings are immutable objects.	

Signature/Header	Example

String toUpperCase()	String s1 = "What ?";
	s1 = s1.toUpperCase();
	System.out.println(s1);

The above example prints WHAT ? This method did not change s1. It returned a reference to a new string that was assigned to s1.	

Signature/Header	Example

String trim()
removes leading and trailing pieces
	String s1 = " a b "; *
	System.out.println(s1.length());
	s1 = s1.trim();
	System.out.println(s1.length());

The above example prints 9 and then 3.	

A Few Last Topics.

(1) The term state refers to the data in an object. The state of a String object refers to the characters in the String. The String class has no mutator methods. Strings are immutable. For example, toUpperCase does NOT make all the letters in a String uppercase; toUpperCase stores the original string changed to upper class characters in a new string! if you use the same name as in the last example*, the original string is overwritten!

(2) String literals are String objects. For example:

```
int n = "hi".length();
```

(3) The primitive type *char* represents a single character. It is not on the AP exam but can be useful. For example:

note, you use single quotes.

concatenation

```
char letter = 'a';
String word1 = letter + "ok";        // this compiles
String word2 = letter + letter;      // this does not compile
```

(4) Some useful escape characters are: \n new line, \t tab over (indent), \" prints "

```
String s = "Say\n\"bye\"";
System.out.println( s );
System.out.println( s.length() );
```