CS 113 – Computer Science I

Lecture 3 – Booleans, Conditionals

Adam Poliak

09/08/2022

Announcements

- Assignment 00
  - Survey
  - Fortune.java
    - Submit on dropbox.
    - Note: make sure to name your file "Fortune.java" (case sensitive)
    - Due tonight
- Assignment 01
  - Due Thursday 09/15
  - Available online

Agenda

- Announcements
- Booleans
- Conditionals

Exercise: Generating a random integer between minVal and maxVal

```java
public static void main(String[] args) {

    int minVal = 2; // inclusive

    int maxVal = 22; // exclusive

    int range = maxVal - minVal;

    double decimalVal =  (Math.random() * range);

    int integerVal = Math.floor(decimalVal);

    int val =  integerVal + minVal;

    System.out.println(val);

}
```

Suppose Math.random() returns 0.2.
What is the output of this program?

Conditionals

A new data type: Booleans

- Contains two possible values:
  - `true; false;`

  - `bool isWet = true;`

- Conditional expression

Conditional Expressions & Relational Operators

- Conditional expression produces either `true` or `false`

- Relational Operators:
    - `>`
    - `>=`
    - `<`
    - `<=`
    - `==`
    - `!=`
- Watch out about `==` vs `=`

Exercise: relational expressions

int temp = 68;

double val = 10.5;

boolean raining = true;

Expression	Value	Type
temp > 80		
val != 5.6		
val >= 10.1		
raining == true		
raining		
raining == false		

Logical Operators

- Way to combine Boolean expressions

- logical Operators:
    - && - and

    - || - or

    - ! - not

Rules of logical operators

1. X && Y is true when
   1. Both X and Y are true

2. X || Y is true when
   1. X is true or Y is true

3. !X is true when
   1. X is false

4. !X false when
   1. X is true

Exercise: logical expressions

boolean isHappy = true;

boolean knowIt = false;

int temp = 40;

| Expression	Value	Type
isHappy && knowIt		
isHappy		
isHappy || temp > 80		
isHappy || knowIt		
!knowIt		
isHappy && (temp < 80 || !knowIt)		

Decision making: if/else

Idea: Branching decision-making based on Boolean expressions

- Example: A **decision tree** for Happy.java

```java
if (isHappy && knowIt) {
    System.out.println("Clap your hands!");

} else {
    System.our.println("Sit quietly.");
}
```

isHappy && knowIt

True

False

"Clap your hands"

"Sit Quietly"
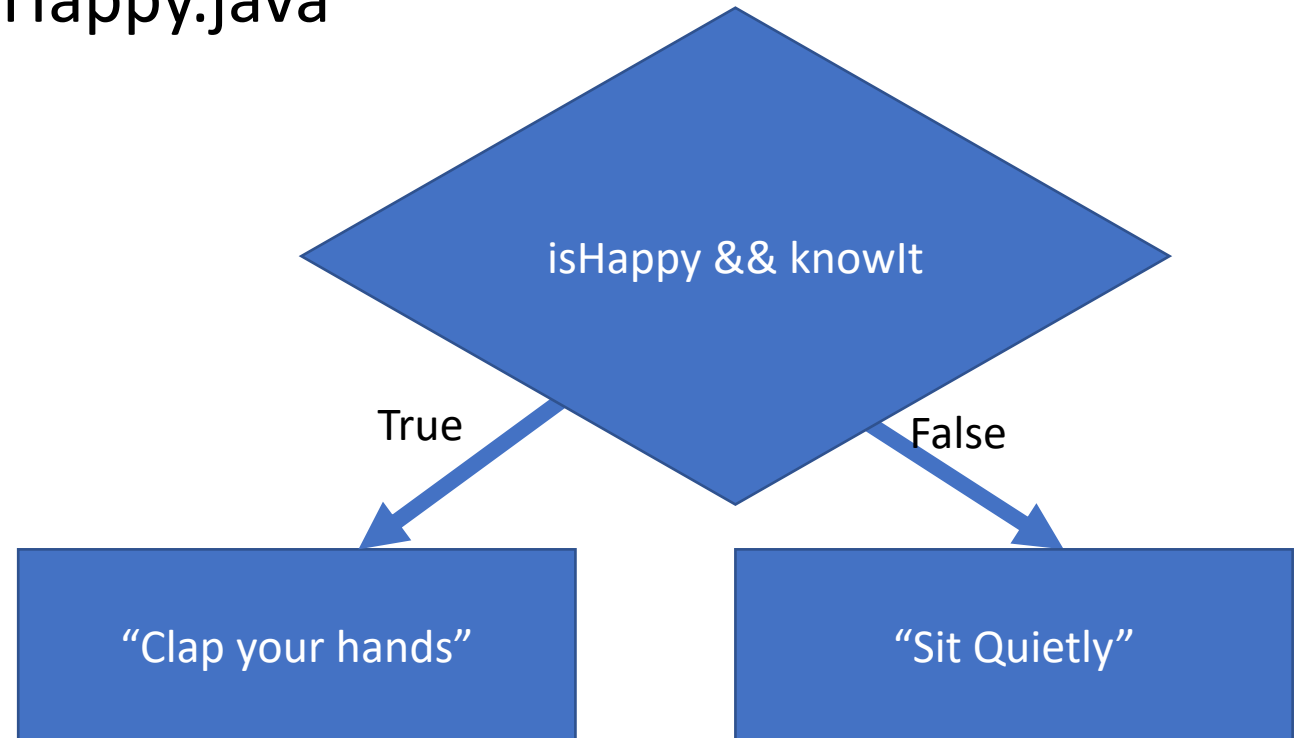
Exercise: IsEven

Write a program IsEven which asks the user for an integer and prints whether it is even or not

```
$ java IsEven
Enter an integer: 4
4 is even!

$ java IsEven
Enter an integer: -1
-1 is odd!

$ java IsEven
Enter an integer: 0
0 is even!
```

Decision making: multi-way if statements

```
if (<condition1>) {
  <stmts>
} else if (<condition2>) {
  <stmts>
}
….
else {
  <stmts>
}
```

NOTES:
- Conditions evaluated in order
- First true condition executes
- Only **one** of the conditions can execute!
- the final else statement is optional

Example: Height.java

- Write a program (called Height.java) that determines if a user can ride a rollercoaster.

- Make sure to ask the user for height in inches.

- Prints out a message if they are taller than 5, 4, 3 feet or are too short for the ride

Exercise: Height.java

```java
class CheckHeight2 {
  public static void main(String[] args) {
    System.out.print("Enter a height (inches): ");
    int h = Integer.parseInt(System.console().readLine());

    if (h > 36) {
      println("Taller than 3 ft");
    }
    else if (h > 60) {
      println("Taller than 5 ft");
    }
    else if (h > 48) {
      println("Taller than 4 ft");
    }
    else {
      println("Too small for this ride");
    }
  }
}
```
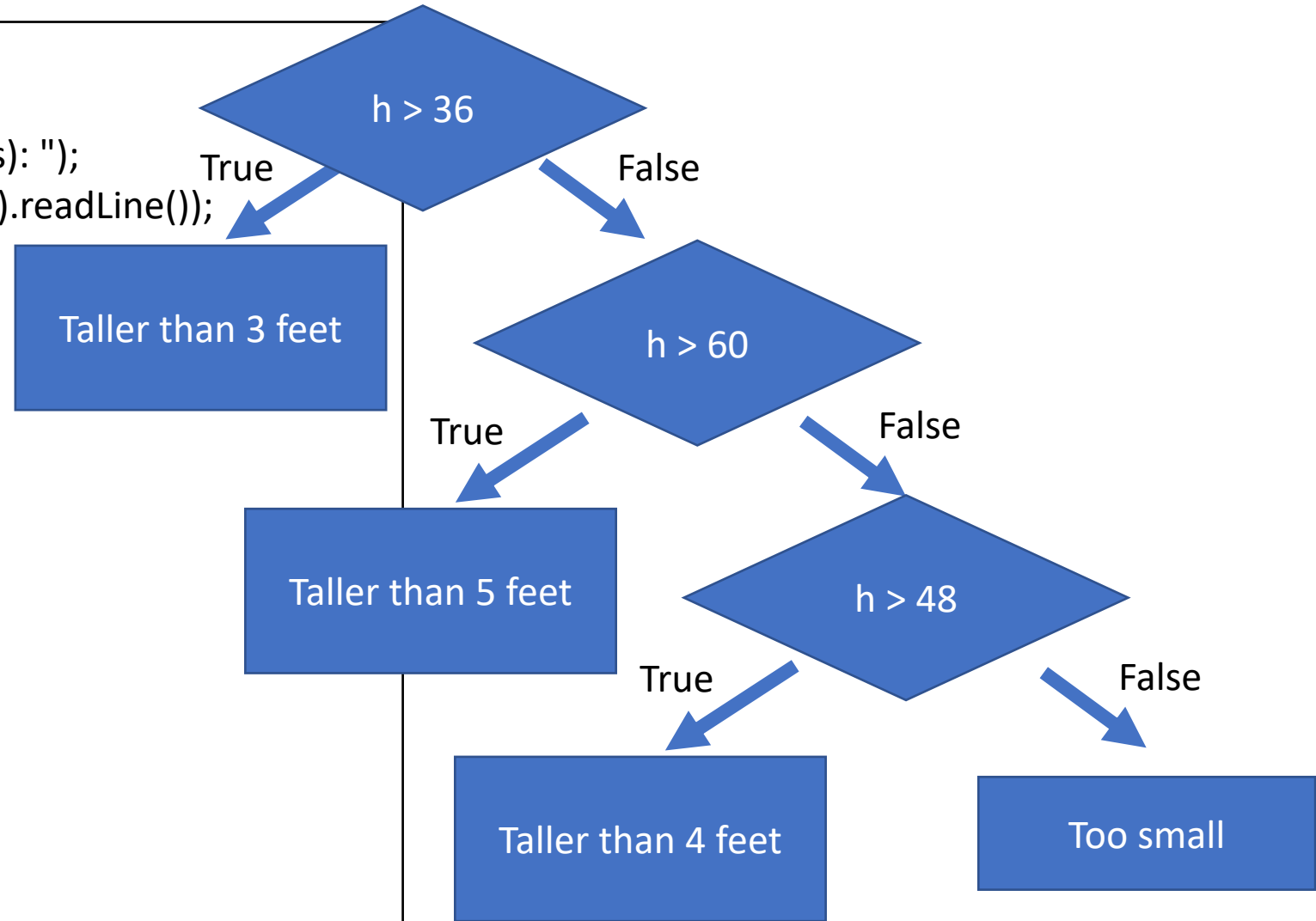
What is the output of this program:

- if the user enters 62 inches?

- if the user enters 10 inches?

Draw the decision tree for this if statement

Exercise: Height.java

```java
class CheckHeight2 {
  public static void main(String[] args) {
    System.out.print("Enter a height (inches): ");
    int h = Integer.parseInt(System.console().readLine());

    if (h > 36) {
      println("Taller than 3 ft");
    }
    else if (h > 60) {
      println("Taller than 5 ft");
    }
    else if (h > 48) {
      println("Taller than 4 ft");
    }
    else {
      println("Too small for this ride");
    }
  }
}
```

Exercise: Height.java

Draw the decision tree for this if statement

What is the output of this program

- if the user enters 62 inches?
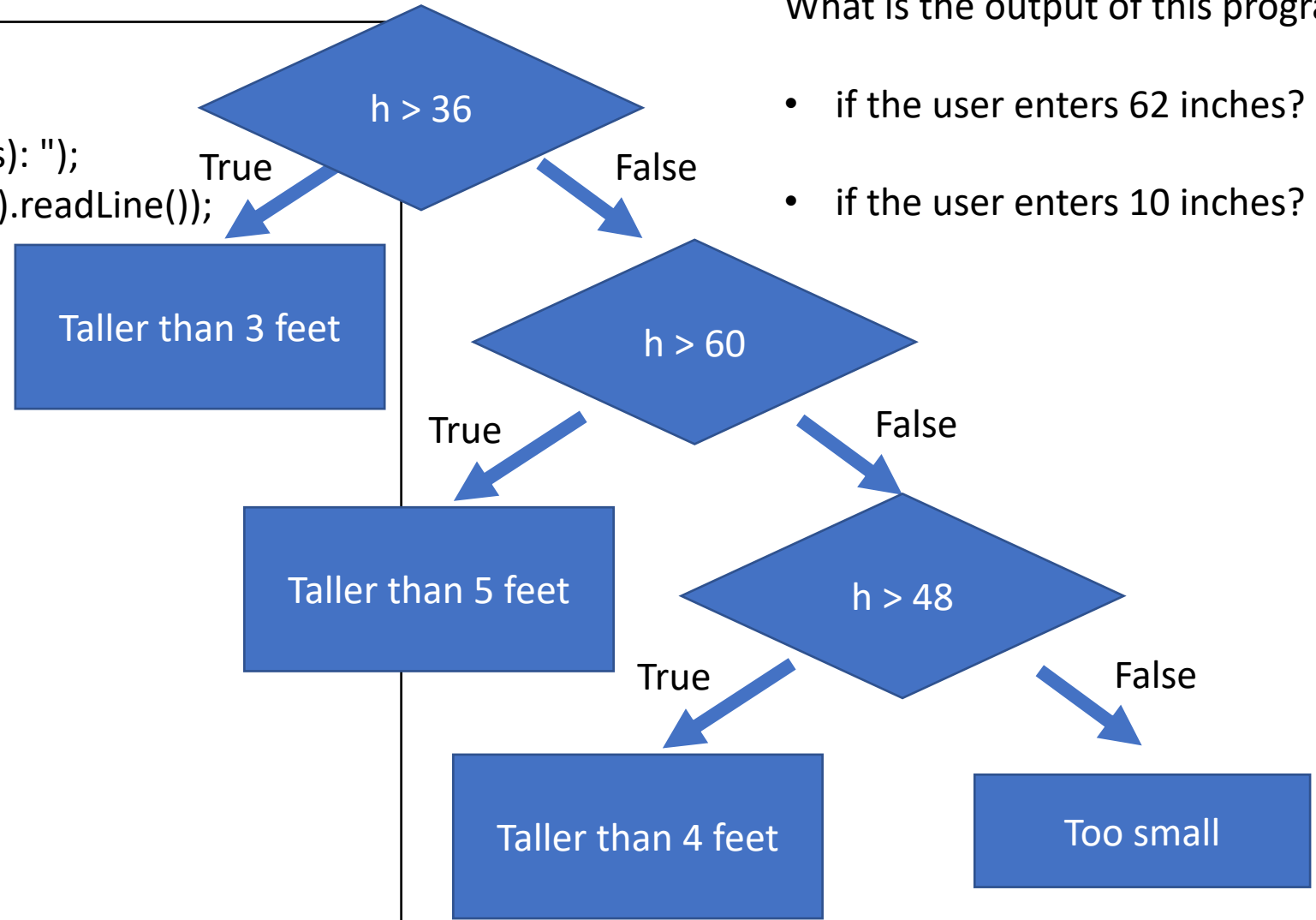
- if the user enters 10 inches?

```java
class CheckHeight2 {
  public static void main(String[] args) {
    System.out.print("Enter a height (inches): ");
    int h = Integer.parseInt(System.console().readLine());

    if (h > 36) {
      println("Taller than 3 ft");
    }
    else if (h > 60) {
      println("Taller than 5 ft");
    }
    else if (h > 48) {
      println("Taller than 4 ft");
    }
    else {
      println("Too small for this ride");
    }
  }
}
```

h > 36

True → Taller than 3 feet

False → h > 60

True → Taller than 5 feet

False → h > 48

True → Taller than 4 feet

False → Too small

Exercise: Blackjack

Write a program Blackjack.java which generates a random value between 2 and 21

- If the value is 21, print the value and "Blackjack" to the console
- If the value is between 17 and 20, print the value and "Stand" to the console
- If the value is less than 17, print the value and "Hit me!" to the console

Style

- How we format our programs is **very** important
  - Like rules of etiquette around eating and keep a clean appearance
  - Like punctuation rules, it helps make text more readable

- Variable names should be descriptive

- Indentation is **very** important
  - Every statement inside a pair of braces must be indented

- Braces should be placed consistently

In VS Code, you can automatically reformat your code!
- Open the command palette: Ctrl-Shift-P
- Search for "reindent"

Comparing strings

- In Java, you cannot directly compare strings: use **compareTo**
  - Javadocs: https://docs.oracle.com/javase/7/docs/api/java/lang/String.html

compareTo

```
public int compareTo(String anotherString)
```

Compares two strings lexicographically. The comparison is based on the Unicode value of each character in the strings. The character sequence represented by this `String` object is compared lexicographically to the character sequence represented by the argument string. The result is a negative integer if this `String` object lexicographically precedes the argument string. The result is a positive integer if this `String` object lexicographically follows the argument string. The result is zero if the strings are equal; `compareTo` returns 0 exactly when the `equals(Object)` method would return `true`.

This is the definition of lexicographic ordering. If two strings are different, then either they have different characters at some index that is a valid index for both strings, or their lengths are different, or both. If they have different characters at one or more index positions, let *k* be the smallest such index; then the string whose character at position *k* has the smaller value, as determined by using the < operator, lexicographically precedes the other string. In this case, `compareTo` returns the difference of the two character values at position `k` in the two string -- that is, the value:

```
this.charAt(k)-anotherString.charAt(k)
```

If there is no index position at which they differ, then the shorter string lexicographically precedes the longer string. In this case, `compareTo` returns the difference of the lengths of the strings -- that is, the value:

```
this.length()-anotherString.length()
```

Specified by:

> `compareTo` in interface `Comparable<String>`

Parameters:

> `anotherString` - the `String` to be compared.

Returns:

> the value 0 if the argument string is equal to this string; a value less than 0 if this string is lexicographically less than the string argument; and a value greater than 0 if this string is lexicographically greater than the string argument.

compareTo

```
public int compareTo(String anotherString)
```

Compares two strings lexicographically. The comparison is based on the Unicode value of each character in the strings. The character sequence represented by this `String` object is compared lexicographically to the character sequence represented by the argument string. The result is a negative integer if this `String` object lexicographically precedes the argument string. The result is a positive integer if this `String` object lexicographically follows the argument string. The result is zero if the strings are equal; `compareTo` returns `0` exactly when the `equals(Object)` method would return `true`.

This is the definition of lexicographic ordering. If two strings are different, then either they have different characters at some index that is a valid index for both strings, or their lengths are different, or both. If they have different characters at one or more index positions, let *k* be the smallest such index; then the string whose character at position *k* has the smaller value, as determined by using the < operator, lexicographically precedes the other string. In this case, `compareTo` returns the difference of the two character values at position `k` in the two string -- that is, the value:

```
this.charAt(k)-anotherString.charAt(k)
```

If there is no index position at which they differ, then the shorter string lexicographically precedes the longer string. In this case, `compareTo` returns the difference of the lengths of the strings -- that is, the value:

```
this.length()-anotherString.length()
```

Specified by:

compareTo in interface Comparable<String>

Parameters:

anotherString - the String to be compared.

Returns:

the value `0` if the argument string is equal to this string; a value less than `0` if this string is lexicographically less than the string argument; and a value greater than `0` if this string is lexicographically greater than the string argument.

```
public int compareTo(String anotherString)
```

Parameters:

anotherString - the String to be compared.

Returns:
- the value 0 if the argument string is equal to this string;
- a value less than 0 if this string is lexicographically less than the string argument;
- and a value greater than 0 if this string is lexicographically greater than the string argument.

Comparing strings

- In Java, you cannot directly compare strings: use **compareTo**

```
String a = "apple";
String b = "banana";
if (a.compareTo(b) == 0) {
    System.out.println("a and b match!");
}
if (a.compareTo(b) != 0) {
 System.out.println("a and b DO NOT match!");
}
```

Lexicographic Values/Order

- Strings are **ordered lexicographically**

  - Generally, the same order as alphabetical order, with some caveats

  - The characters of a string each correspond to a number

ASCII

Dec	Hx	Oct	Char		Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL	(null)	32	20	040	 	Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH	(start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX	(start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX	(end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT	(end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ	(enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK	(acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL	(bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS	(backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB	(horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF	(NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT	(vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF	(NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR	(carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO	(shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI	(shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE	(data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1	(device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2	(device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3	(device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4	(device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK	(negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN	(synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB	(end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN	(cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM	(end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB	(substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC	(escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS	(file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174	|	|
29	1D	035	GS	(group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS	(record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US	(unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

https://www.asciitable.com/

StringCompare.java

```java
String first = "a";
String second = "A";
int asciia = (int) first.charAt(0);
int asciib = (int) second.charAt(0);
System.out.println("ASCII Code for "+first+" is " + asciia);
System.out.println("ASCII Code for "+second+" is " + asciib);

if (first.compareTo(second) == 0) {
    System.out.println(first+" is equal to "+second);
}
else if (first.compareTo(second) < 0) {
    System.out.println(first+" is less than "+second);
}
else if (first.compareTo(second) > 0) {
    System.out.println(first+" is greater than "+second);
}
```

```
$ java StringCompare
ASCII Code for a is 97
ASCII Code for A is 65
a is greater than A
```

Exercise: IsPrimary

Write a program that asks the user for a color and prints whether the color is primary or not.

- The primary colors are "red", "green", "blue"

- All other inputs are non-primary

$ **java IsPrimary**
Enter a color: **green**
green is not primary

$ **java IsPrimary**
Enter a color: **blue**
blue is primary