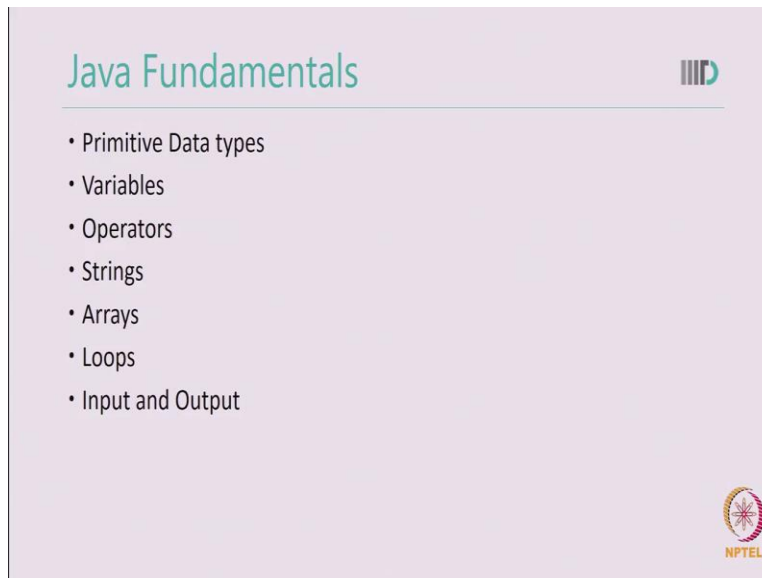


**Mobile Computing**  
**Professor Pushendra Singh**  
**Indraprastha Institute of Information Technology Delhi**  
**Java Basics**  
**Lecture 02**

Hello, in this lecture we will learn about some fundamentals concepts of java.

(Refer Slide Time: 0:16)



We will cover primitive data types, variables, operators, strings, arrays, loops and basic input and output.

(Refer Slide Time: 0:32)

## Primitive data types



- Integer types
  - int, short, long, byte
- Floating-point types
- Char
- Boolean



Primitive data types in java. Though java is a object oriented language it also provide some primitive data types. Basic types such as int, short, long, byte which you may have also used in C or C plus plus program are also available in java. You need not to create an object of this type. They are available to you by the java programming language.

(Refer Slide Time: 1:00)

## Integer types



Type	Size	Range
int	32 bits	
short	16 bits	
long	64 bits	
byte	8 bits	



In java an integer is of 32 bits short 16 along 64 and byte 8 bits. Irrespective of the platform the length remains the same. This is very different than a C plus plus where the length depends on the hardware platform that you are using. This makes java program is easier to run on different types of hardware and you know what type of behavior you will expect from a java program.




(Refer Slide Time: 1:29)

## Floating-point types

Type	Size	Range
float	32 bits	$\pm 3.40282347E+38$
double	64 bits	$\pm 1.79769313486231570E+308$

- float has a suffix F
  - By default any number having decimal is double
- BigDecimal: for high precision




Similarly a floating point is of size 32 and a double is of size 64. I have given the range of floating point and the double here. And I ask you to find out the range of int, short and byte yourself. A float has a suffix F and we can also use big decimal for height precision in java.

(Refer Slide Time: 1:53)

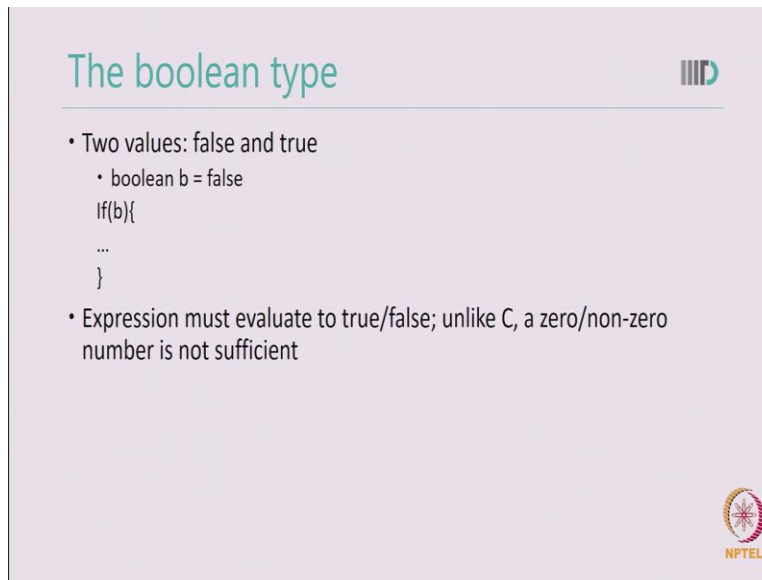
## The char type

- *char* describes individual characters
  - 'A', 'B', '['
    - Unicode values
- A *char* is a code unit of UTF-16




The java character type is similar to the character type of C or C plus plus it uses a single code and uses unique code values. The java character is the code unit of UTF 16.

(Refer Slide Time: 2:10)



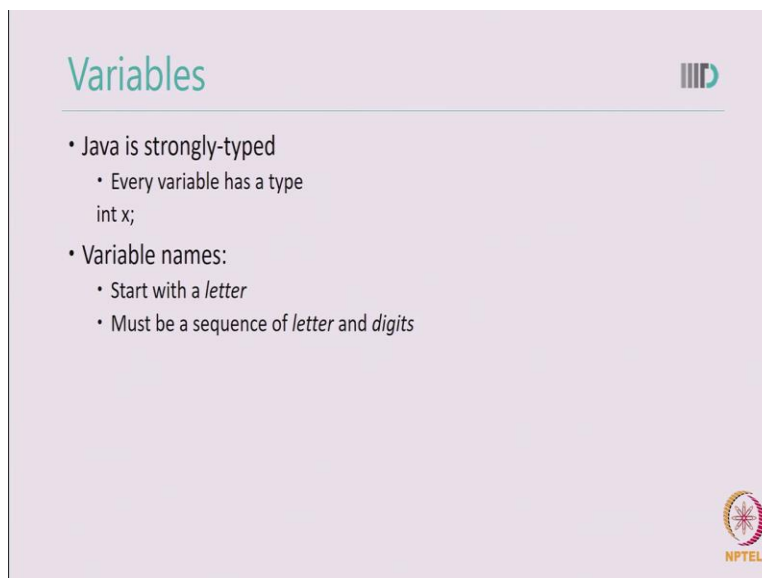
## The boolean type

- Two values: false and true
  - `boolean b = false`
- ```
if(b){  
...  
}
```
- Expression must evaluate to true/false; unlike C, a zero/non-zero number is not sufficient



Java Boolean has two values false and true. In java an expression must evaluate to true or false. This is different than a C or C plus plus where a zero or non zero value works. For example you are using an if loop in java the expression must be a Boolean expression just a zero will not work.


(Refer Slide Time: 2:32)



## Variables

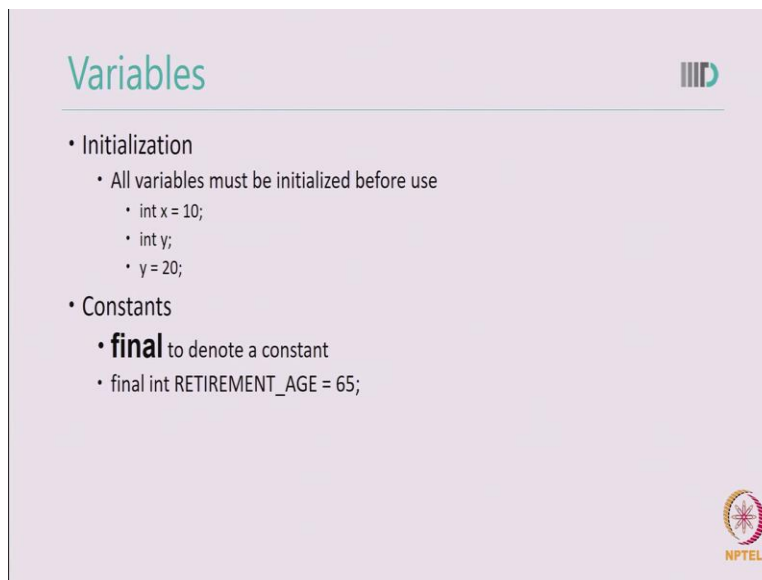
- Java is strongly-typed
  - Every variable has a type

```
int x;
```
- Variable names:
  - Start with a *letter*
  - Must be a sequence of *letter and digits*




Java is a strongly typed language which means that each variable must have a type and each variable name must has start with a letter. After the initial letter it may include letter or other digits.

(Refer Slide Time: 2:50)



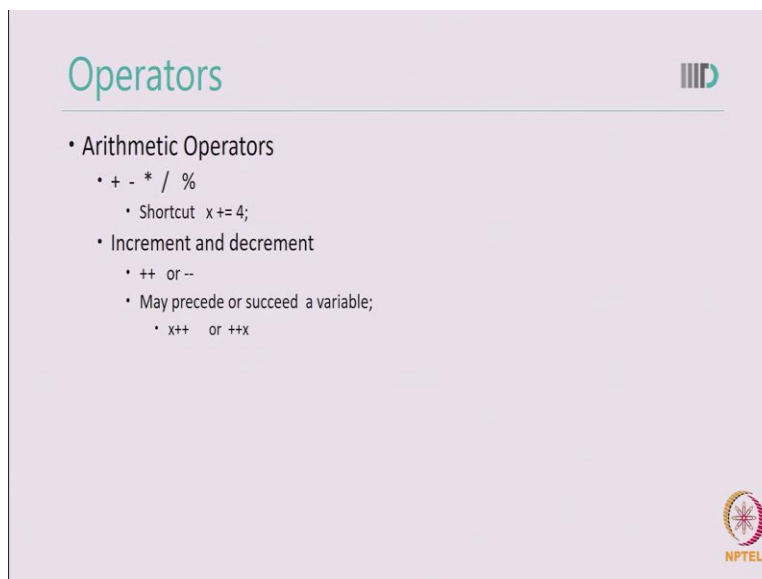
## Variables

- Initialization
  - All variables must be initialized before use
    - `int x = 10;`
    - `int y;`
    - `y = 20;`
- Constants
  - **final** to denote a constant
  - `final int RETIREMENT_AGE = 65;`




All variables must also be initialized before use. Java automatically initializes these variables for you. However my advice is to initialize them yourself so that you know what values these variables hold. You may define a variable as a final that means the value of that variable cannot be changed and it acts like a constant. As told earlier for such variables we use all capital letters. There is a simple example of a variable called retirement age which is set to 65 and is declared as final.

(Refer Slide Time: 3:29)



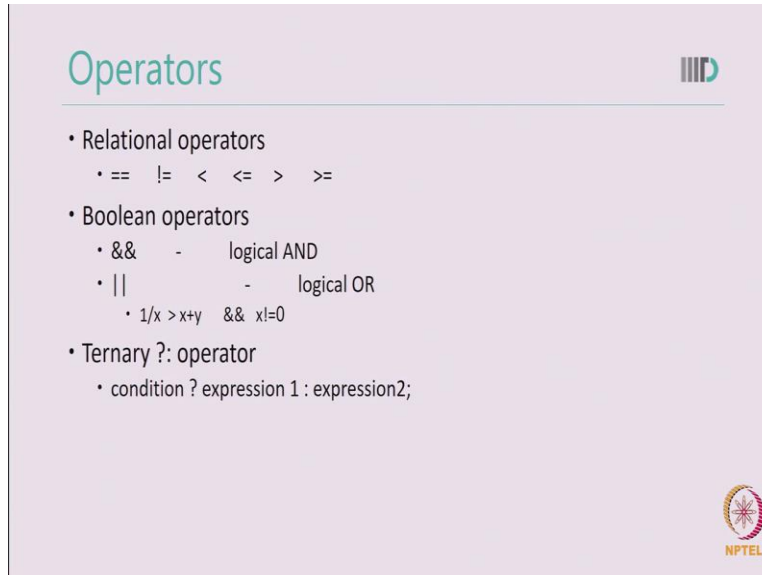
## Operators

- Arithmetic Operators
  - `+` `-` `*` `/` `%`
  - Shortcut `x += 4;`
- Increment and decrement
  - `++` or `--`
  - May precede or succeed a variable;
    - `x++` or `++x`



Just like c or c plus plus java gives you all the operators plus, minus, multiplication, division, Etc and other increment and decrement operators.

(Refer Slide Time: 3:42)

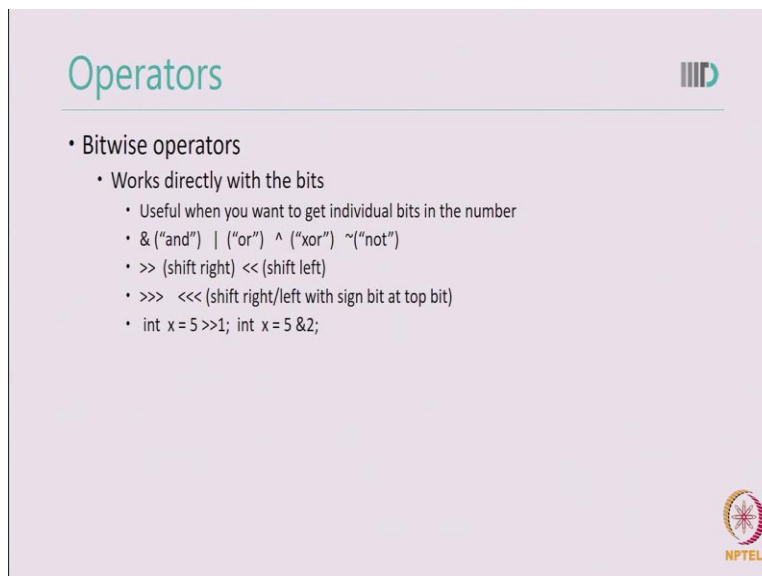


The slide is titled "Operators" and features a list of operator categories. It includes a logo in the top right and an NPTEL logo in the bottom right.

- Relational operators
  - == != < <= > >=
- Boolean operators
  - && - logical AND
  - || - logical OR
    - 1/x > x+y && x!=0
- Ternary ?: operator
  - condition ? expression 1 : expression2;

Other relational, Boolean and ternary operators just like C or c plus plus. I am assuming that you are already programmed in c or c plus plus so I am not going into the details of this operators works.

(Refer Slide Time: 3:56)



The slide is titled "Operators" and features a list of bitwise operators. It includes a logo in the top right and an NPTEL logo in the bottom right.


- Bitwise operators
  - Works directly with the bits
    - Useful when you want to get individual bits in the number
    - & ("and") | ("or") ^ ("xor") ~("not")
    - >> (shift right) << (shift left)
    - >>> <<< (shift right/left with sign bit at top bit)
    - int x = 5 >>1; int x = 5 &2;

Java also provides you bitwise operators which you can use if you want to work directly with the bits.

(Refer Slide Time: 4:04)

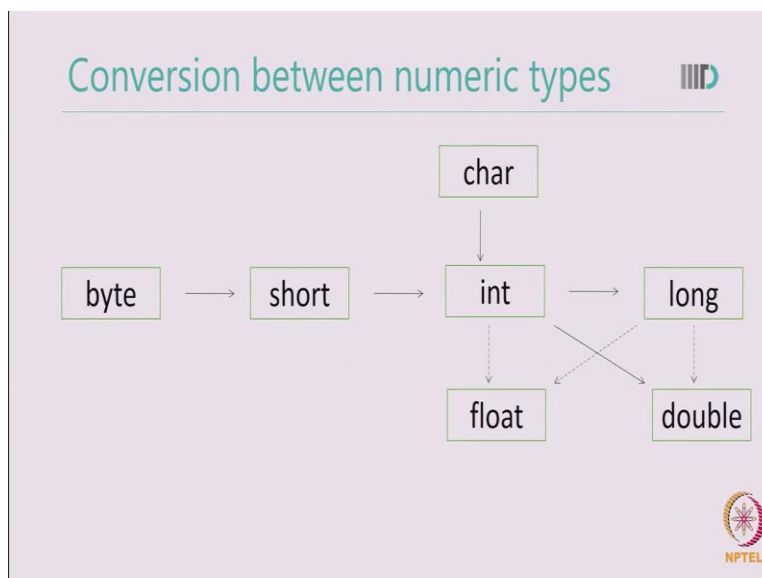
## Mathematical Functions

- Useful Mathematics functions
  - `Math.sqrt(double x);`
  - `Math.pow(x, y)`
  - `Math.sin, Math.cos...`
  - `Math.exp, Math.log`
  - Mathematics constants
    - `Math.PI, Math.E`



Java library also provides you access to different mathematical functions. For example square root, power of, sin, cos, exponential, log functions and many others. Please use java documentation to know what mathematical functions are available for java

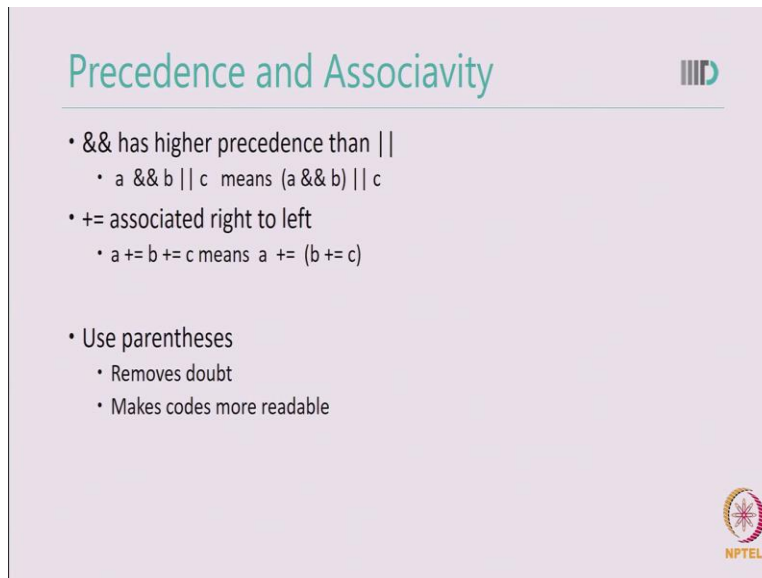
(Refer Slide Time: 4:21)





Java allows you conversion from one numeric type to another as you may see that if you are converting from smaller numeric type to another bigger numeric type then you may not lose information but if you are doing the other way down that you may lose information.

(Refer Slide Time: 4:38)



The slide is titled "Precedence and Associativity" and features a logo in the top right corner. It contains the following text:

- `&&` has higher precedence than `||`
  - `a && b || c` means `(a && b) || c`
- `+=` associated right to left
  - `a += b += c` means `a += (b += c)`
- Use parentheses
  - Removes doubt
  - Makes codes more readable

In the bottom right corner, there is a logo for NPTEL (National Programme on Technology Enhanced Learning).


Precedence and associativity, you may have learnt about that in your other programming language like C or C plus plus as well. You may either choose to remember all the rules about precedence or associativity, however the recommended uses to use the parenthesis. Parenthesis removes the doubt and make your code more readable. For example if you are using `A plus equal to B plus equal to C` using the precedence and associativity it will be calculated as first `B plus equal to C` and then `A plus equal to B` expression that has to be calculated. The best way to use the parenthesis so that it is clear to the programmer. What do you mean without having to go into the details of precedence and associativity.

(Refer Slide Time: 5:31)

## Enumerated types

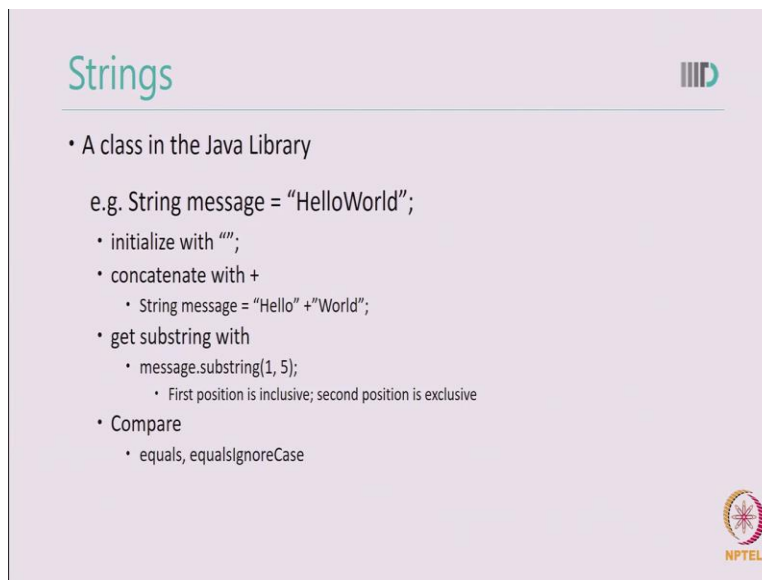
- For making a variable hold restricted values
- You own Pizza hut 😊

```
enum Size {SMALL, MEDIUM, LARGE}
.....
Size s = Size.LARGE;
....
if { s == Size.MEDIUM}{
System.out.println("Medium ordered");
}
else{
System.out.println("Medium not ordered");
}
```



Java also gives you enumerated types. Which means that you can declare your own restricted values. Suppose you own the pizza hut and that's why you would like to describe the size as a small, medium or large. In (5:44) type allows you that possibility. So you can declare your own data such as a size which has three values of a small, medium and large. Then you can refer to them using the dot operator. You can use them as any other types. For example you may have an if else block which has S comparing to size medium or comparing it other sizes that you have described.

(Refer Slide Time: 6:13)



The slide is titled "Strings" in a teal font at the top left. In the top right corner, there is a logo consisting of three vertical bars of increasing height followed by a stylized 'D'. Below the title, there is a list of bullet points:

- A class in the Java Library

Below the list, there is an example of code: `e.g. String message = "HelloWorld";`

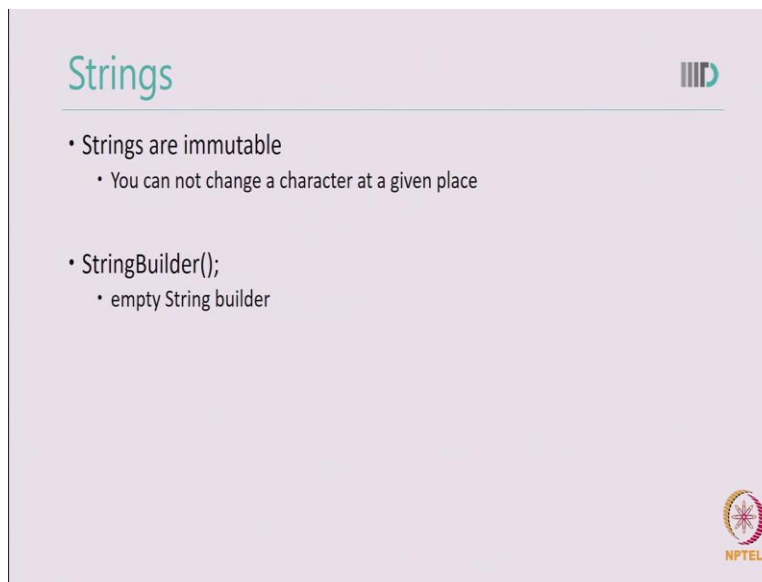
- initialize with "";
- concatenate with +
  - `String message = "Hello" + "World";`
- get substring with
  - `message.substring(1, 5);`
    - First position is inclusive; second position is exclusive
- Compare
  - `equals, equalsIgnoreCase`

In the bottom right corner, there is a circular logo with a star-like pattern and the text "NPTEL" below it.

Strings, Java has a class and java library called as strings that allows you different strings functionality that you may need for your programs. A string is initialized with double quotes as you can see in the examples. I am having a variable called message of the type strings which is initialized to the value of helloworld. I can concatenate in a different strings using the plus. I can also get sub string using a method called substring. Please note that the first position is an inclusive but the second position is not inclusive also in java strings start counting from zero.

So if I give 1 to 5 I actually mean the second letter and not the first letter. In the given example of 1 to 5 I will actually have E L L and O as a result of the message dot substring call. If you want to compare two strings the advisable method is to use equals and equals ignore case. Equals method compares the value held by a string variable.

(Refer Slide Time: 7:30)

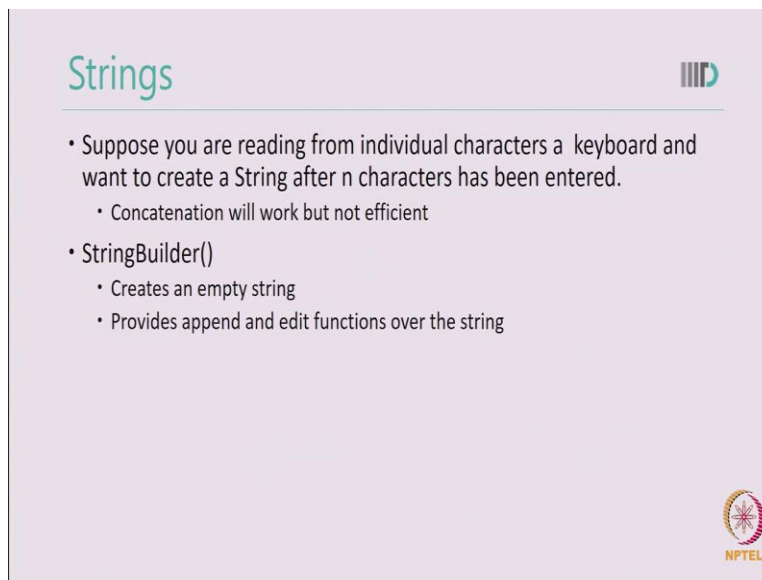


The slide is titled "Strings" in a teal font at the top left. In the top right corner, there is a logo consisting of three vertical bars of increasing height followed by a stylized 'D' shape. Below the title, there are two bullet points: the first is "Strings are immutable" with a sub-bullet "You can not change a character at a given place"; the second is "StringBuilder();" with a sub-bullet "empty String builder". In the bottom right corner, there is a circular logo with a star-like pattern and the text "NPTEL" below it.

Strings are immutable in java. That is once you have created a string you cannot change it and JVM keeps that variable in the memory as long as your program lives. If you want to create mutable strings please use string builder as your base class and not the java.

For checking for equality do not use double equal to sign. Double equal to sign may introduce intermittent bugs which you may not know. Later on we will see a program which will show you a comparison of a strings using double equal to sign and using equals method. To demonstrate you how the double equals and equals method works. Whenever you want to check equality of two strings always use equals or equals ignore case. Do not ever use double equal to sign. There are different methods which (all) which java provides you for example length return you the length of the strings. You can access individual character by using character act.

(Refer Slide Time: 8:34)



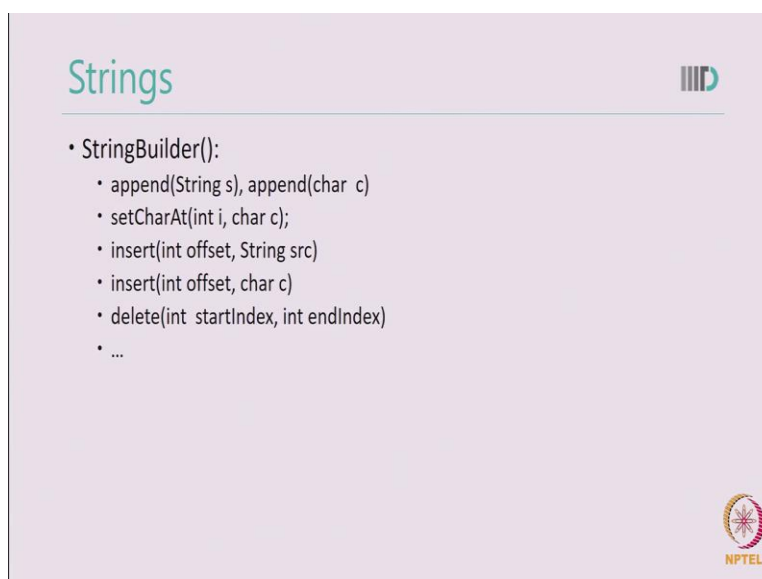
The slide is titled "Strings" in a teal font at the top left. In the top right corner, there is a logo consisting of three vertical bars of increasing height followed by a stylized 'D'. Below the title, there is a horizontal line. The main content is a bulleted list:

- Suppose you are reading from individual characters a keyboard and want to create a String after n characters has been entered.
  - Concatenation will work but not efficient
- `StringBuilder()`
  - Creates an empty string
  - Provides append and edit functions over the string

In the bottom right corner, there is a circular logo with a star-like pattern and the text "NPTEL" below it.

Suppose you are reading from individual characters from a keyboard and you want to create a string in this case you may use a concatenation. However as you know that java strings are immutable the longer you use concatenate java will create those many strings into the virtual memory. This is not very efficient. For such cases you should ideally use a string builder which provides append and edit functions over the strings.

(Refer Slide Time: 9:06)



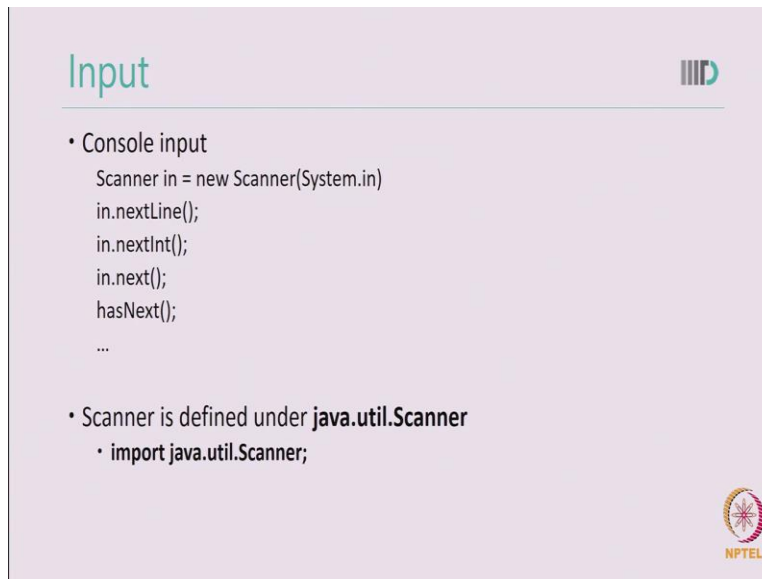
The slide is titled "Strings" in a teal font at the top left. In the top right corner, there is a logo consisting of three vertical bars of increasing height followed by a stylized 'D'. Below the title, there is a horizontal line. The main content is a bulleted list:

- `StringBuilder()`:
  - `append(String s), append(char c)`
  - `setCharAt(int i, char c);`
  - `insert(int offset, String src)`
  - `insert(int offset, char c)`
  - `delete(int startIndex, int endIndex)`
  - ...

In the bottom right corner, there is a circular logo with a star-like pattern and the text "NPTEL" below it.

Here are some methods which are provided by string builder class. You may want to write a program using a string and a string builder and check these different functionalities by yourself.

(Refer Slide Time: 9:21)



The slide is titled "Input" in a teal font at the top left. In the top right corner, there is a logo consisting of three vertical bars of increasing height followed by a stylized 'D' shape. The main content of the slide is a list of bullet points. The first bullet point is "Console input", followed by a code block: 


```
Scanner in = new Scanner(System.in)
in.nextLine();
in.nextInt();
in.next();
hasNext();
...
```

 The second bullet point is "Scanner is defined under **java.util.Scanner**", with a sub-bullet point "import java.util.Scanner;". In the bottom right corner, there is the NPTEL logo, which is a circular emblem with a star-like pattern inside, and the text "NPTEL" below it.


Input and output, java provides us different ways of taking input and also different ways to print output. In this lecture we will only discuss the basic input method that you may immediately need. One such method is to use an object of the type scanner. A scanner is defined under java dot util dot scanner class. In order to use a scanner you will have to do an import of java dot util dot scanner class. A scanner can be used as shown here by declaring a variable of type scanner you can initialize it. After that you can use different methods that are available to you such as nextline, nextint or simply next. You can also use methods such as has next to find out if there is anything left (to) to scan.

(Refer Slide Time: 10:20)

## Formatting Output




- Since Java 5.0
  - `System.out.printf()` – just like in C
  - e.g.
  - `float x = 3333.33333f;`
  - `System.out.printf("%8.2f", x),`
  - `System.out.printf("%.2f", x)`
- For Strings
  - `String.format=(string, args);`
  - e.g.
  - `String name = "Ram";`
  - `String s = String.format("hello %s", name);`
  - `System.out.print("%tc", new Date());`




Output, just like c or c plus plus java allows you multiple ways to format your output. Here are some simple examples given for example you may want to print an output of the type where you display only two decimal values. Java provides many other output opportunities and I advised you to refer the java documentation. To know about different output functionalities that are provided by java.

(Refer Slide Time: 10:48)

## File input and Output



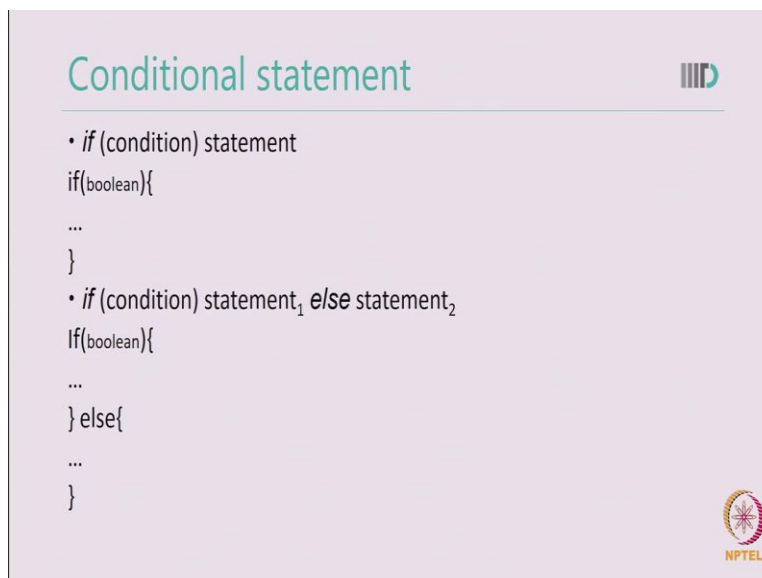
- To read
  - `Scanner in = new Scanner(new File("file.txt"));`
  - Use Scanner functions to read further.
  - If file is not found, exception will be thrown
- To write
  - `PrintWriter out = new PrintWriter("file.txt");`
  - Use all the function as with System.out
  - A new file will be created or overwritten.



Java also allows you to read files which are available on your file system and also to write to the files which are available on your file system. You may want to even create new files from a java program.

You may use the same scanner to read from a file except that this time you have to initialize the scanner with the file that is present on your (f) on your file system. Similarly for writing to a file you may use print writer which is another java class that's available to you.

(Refer Slide Time: 11:23)



The slide is titled "Conditional statement" and features a light purple background. It contains two bullet points, each followed by a code snippet. The first bullet point is "• if (condition) statement" and the code is: `if(boolean){`  
`...`  
`}`. The second bullet point is "• if (condition) statement<sub>1</sub> else statement<sub>2</sub>" and the code is: `If(boolean){`  
`...`  
`} else{`  
`...`  
`}`. In the top right corner of the slide, there is a logo consisting of three vertical bars of increasing height. In the bottom right corner, there is a circular logo with a star-like pattern and the text "NPTEL" below it.

Now, let us come to the conditional statement. I have already used if and else in the beginning of this lecture and you may be familiar with if and else by your prior experience with c or c plus plus or other programming languages. If and else provides you the basic conditional statement essentially if evaluates a Boolean expression and if that expression is true then the commands next to the if block are executed. If those are not true then the else block is executed. You may use if and else together or you may only use if. In java as a stated earlier the expression must be a Boolean and zero or non zero simply doesn't work as they would work in c or c plus plus program.




(Refer Slide Time: 12:12)

## Conditional statement

- if else if

```
if(age <= 5){  
if( boolean){  
} else if(boolean){  
} else if(boolean){  
} else {  
}  
} else if(age <= 10){  
} else if(age <=15){  
} else if(age <=20){  
} else {  
}
```




Here is a simple program of using multiple if and else statement or called nested if and else block. For example you can see I start with an if and then I go else if else if else if and finally I end with an else. You may want to write similar programs for checking for multiple questions.

(Refer Slide Time: 12:41)

## Multiple Selection - Switch

```
Scanner in = new Scanner(System.in)  
int choice = in.nextInt();  
  
switch (choice){  
case 1: ...  
    break;  
case 2: ...  
    break;  
  
default:  
    break;  
}
```

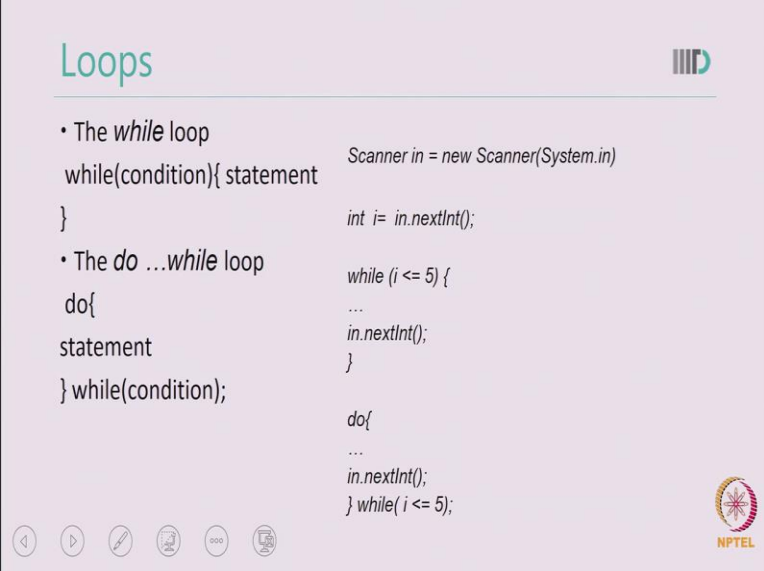
- Same as *if-else-if* but
  - Condition can only be integer or enumerated constants
  - In absence of break, execution falls through to the next alternative



An alternative to use nested if and else is using a switch statement. Switch is same as nested if and else. However the condition can only be an integer or enumerated constants. The syntax of switch is switch given a expression and then you can have multiple cases to execute depending on the expression at it evaluates to. At the end it is mandatory to write break. If you don't write

break then switch simply false through. There is also a default statement which executes if none of the cases are true. You may choose to use nested if else or switch as per your convenience.

(Refer Slide Time: 13:34)




The slide is titled "Loops" and features the NPTEL logo in the top right corner. It contains two bullet points, each with a code example. The first bullet point is "The while loop" and shows a code snippet: `Scanner in = new Scanner(System.in)`, `while(condition){ statement`, and `} int i= in.nextInt();`. The second bullet point is "The do ...while loop" and shows two code snippets: `while (i <= 5) { ... in.nextInt(); }` and `do{ ... in.nextInt(); } while( i <= 5);`. At the bottom of the slide, there are navigation icons (back, forward, search, etc.) and the NPTEL logo.

Let's now come to loops. Just like c or c plus plus java allows you both indeterminate loop and determinate loop. Indeterminate loop is the one in which you do not know how many times a loop will execute. Example of such loop structure are while and do while loops. Let's see a simple program on the right side. As you can see that in the first loop I am using while statement and I am checking that as long as I is smaller than five the loop will keep getting executed now this loop may execute 5 times 1 time, 0 time or even infinite number of times.

In a second case I am using a do while loop which is also checking for the same condition. And essential difference between while and do while is that the do while loop execute at least once. For example no matter what the value of I is the do while loop will definitely execute false. However if I am just using the while loop and the condition doesn't satisfy in the very beginning then the while loop doesn't execute.


(Refer Slide Time: 14:42)

## Determinate Loops



- *For* loop


```
for ( int i = 0; i <=10; i++)  
{  
    System.out.println("I = "+i);  
}
```




The determinate loop example is the for loop. In for loop we know how many times the loop is going to execute. For example in this given for loop I know that this loop will execute for the value of I starting from zero till the value of I reaches ten and I keep on incrementing the value of I for each loop iteration. For loop is very useful when we know how many times the loop will execute. You may have already used for and do while and while loop in your C and C++ programs.

(Refer Slide Time: 15:22)

## Statements that break control flow

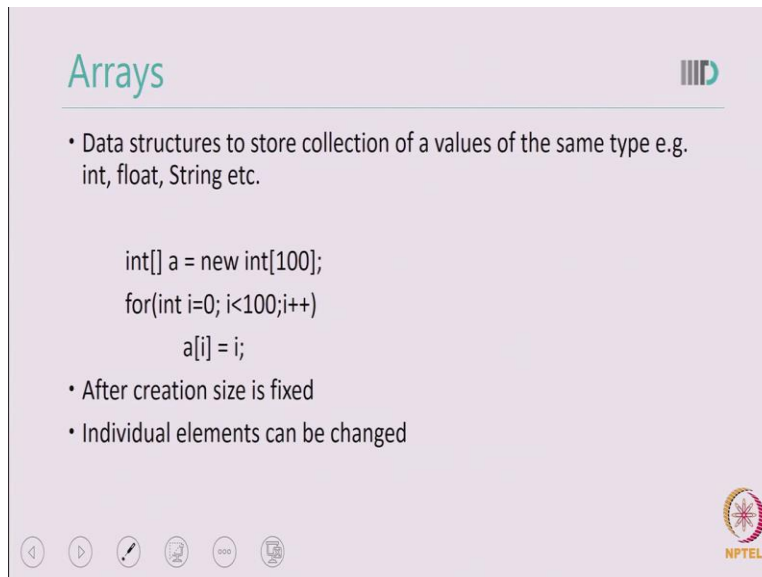


- break
  - Takes you out of loops
- continue
  - Transfers control to the header of innermost loop
- label
  - Allows a block of code to be labeled
  - Can be used with break to jump out of a block of code or nested loop
  - You can not jump into a block



With for and while loop we also sometimes use statement such as break or continue or labeled to break the control flow. Break takes you out of the loop continue transfers the control to the header of the innermost loop and labels allows you a block of code will be labeled which you can jump into. Traditionally use of brake, continue and label is (())(15:47) and you are suppose to use a control flow that is clear to the programmer.

(Refer Slide Time: 15:54)



The slide is titled "Arrays" and features a purple background. It includes a list of bullet points, a code snippet, and a list of additional bullet points. At the bottom, there are navigation icons and the NPTEL logo.

## Arrays

- Data structures to store collection of a values of the same type e.g. int, float, String etc.

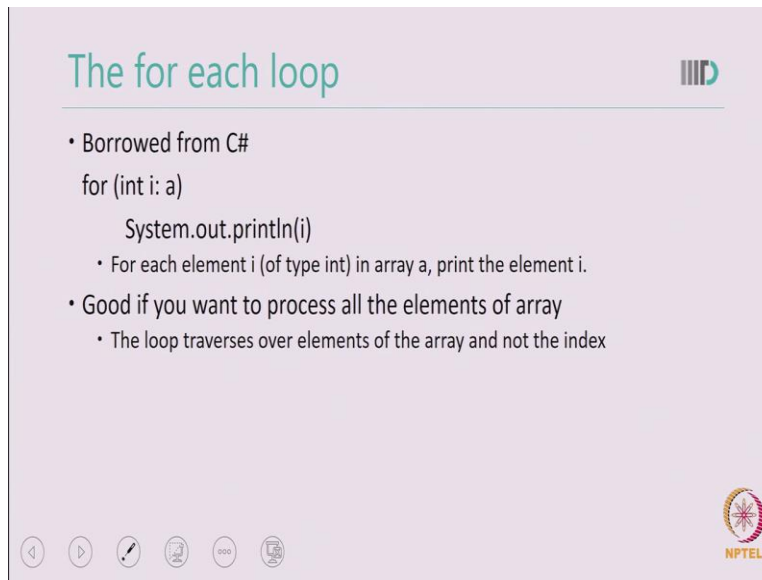
```
int[] a = new int[100];  
for(int i=0; i<100;i++)  
    a[i] = i;
```

- After creation size is fixed
- Individual elements can be changed

NPTEL

As you may already know arrays are data structure to store collection of values of the same type. For example if you want to store age of all the students in your class. In such cases you may want to use an array of type integer to store ages of all the students in single data structure. The syntax for array is as shown here in the example. Here in the given example I am declaring an array of type int and of size hundred. Then I am initializing that array to a given value. The limitations with array is that after creation the size of the array is fixed. However the advantage of array is that individual elements can be changed very easily using the index of the array.

(Refer Slide Time: 16:52)



The slide is titled "The for each loop" in a teal font. In the top right corner, there is a logo consisting of three vertical bars of increasing height. The main content consists of a bulleted list and a code snippet. The code snippet shows a C# for-each loop: `for (int i: a)` followed by `System.out.println(i)`. The first bullet point states "Borrowed from C#" and the second bullet point states "Good if you want to process all the elements of array". A sub-bullet under the second point explains that the loop traverses over elements of the array and not the index. At the bottom of the slide, there are navigation icons (back, forward, search, etc.) and the NPTEL logo in the bottom right corner.

## The for each loop

- Borrowed from C#  
for (int i: a)  
    System.out.println(i)
  - For each element i (of type int) in array a, print the element i.
- Good if you want to process all the elements of array
  - The loop traverses over elements of the array and not the index



A very good way to initialize an array is using a for each loop. For each loop introduced in a language called C sharp you may already be aware of it unlike the for loop, for each loop has a very compact syntax. For example here is a for each loop which will go through an array variable named A. the array A is an integer array I am using a rephrase variable I and I am printing out all the values of the array elements. The for each loop here goes through all the elements of the array A and I can access those elements using the variable I. for each loop is very useful if you want to process all the elements of array it provides you a very compact syntax.

(Refer Slide Time: 17:44)

## Array Initialization

```
int[] numberArray = {1,2,3,4,5,6};
```

- No need to call *new* here and size is also determined by the initialization
- for each loop is good for initialization



Array is can also be initialized directly at the time of creation. For example in this case I am initializing an array to the value 1, 2, 3, 4, 5, 6 in this case I need not to even define the size of an array. The initialization block itself defines the size so this number array the size is six and it is already initialized to 1, 2, 3, 4, 5, and 6. I also need not to call *new* here such initialization is very useful when you have array of a smaller size and when you already know the value that you need to initialize each element with.

(Refer Slide Time: 18:27)

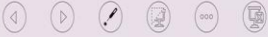

## Array Copying

```
int[] newArray = numberArray
```

- Only copies reference

```
int[] newArray = Array.copyOf(numberArray, length);
```

- Copies the array elements



If you want to copy an array you can use a simple equals sign. However this will work only by copying the reference which means that now the two variables will refer to the same array and the change in the array will be reflected by both variables. If you really want to copy all the elements of an array please use the method called `copyOf`. This copies all the array elements.

(Refer Slide Time: 18:57)



The slide is titled "Common Array Operations" and lists several methods from the `Arrays` class:

- **Sorting**
  - `Arrays.sort(numberArray);`
  - Uses quicksort algorithm
- **Printing**
  - `Arrays.toString(numberArray);`
- **Search**
  - `Arrays.binarySearch(numberArray, 5);`
  - index or -1 is returned
- **Equality**
  - `Arrays.equals(type[] a, type[] b)`
- ...

The slide also features a logo in the top right corner and the NPTEL logo in the bottom right corner.

Java also provides you common array operations sometimes you may want to sort all the array elements. Java provides you a method called `sort` which uses the quick sort algorithm. Similarly you can print all the array elements by using the `toString` method.

You can also do search within the array elements by using a method called `binarySearch` which uses the binary search algorithm to search for an array and if the element is found then the index of that array element is returned and if the element is not found then minus one is written. Similarly Java provides methods for equality. There are many more Java methods which are available for you for array operations. I advised you to refer to Java documentation to learn about array operations.

(Refer Slide Time: 19:52)


## Multidimensional Array

- Multi-dimensional is actually arrays of arrays  

```
int[][] a = new int[rows][columns]
```

  - Single dimension array of rows
  - Each row element is single dimension array of columns
- For each will not work directly
  - Apply on rows and then on columns  

```
for(int[] row : a)  
  for(int i : row)
```
- To print:
  - `Arrays.deepToString(a)`



Besides one dimensional array you may also declare multi dimensional array in java. Multi dimensional arrays work as rows and column. For example if I want to declare a two dimensional array I may want to declare as rows and columns. If I want declare three dimensional arrays I will have to add another element here and so on. For each does not work directly for multi dimensional array. Here I am giving a simple code example to initialize or to access multi dimensional arrays.

You first go through a row and then inside each row you may use for each loop. So each dimension of an array acts like a independent array. If you want to understand it from a for each loop point of view. This is all about arrays in java I advise you to write simple program in java which uses array and revised your concept of arrays. Java library provides you various methods (to) which directly works with array. This unlike c or c plus plus where you may have to write many of these methods yourself. Thank you! In the next lecture we will discuss about object and classes.