Scanner
String Chopping

Scanner Methods

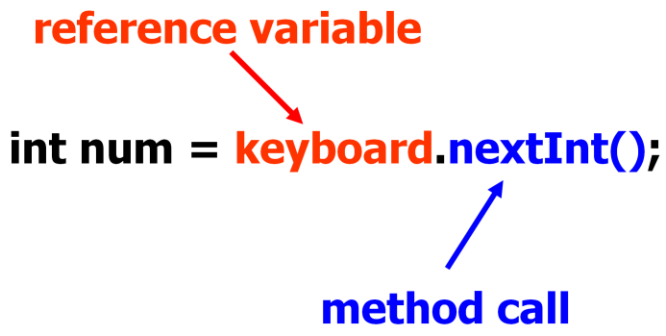
Scanner
frequently used methods

Name	Use
nextInt()	returns the next int value
nextDouble()	returns the next double value
next()	returns the next one word String
nextLine()	returns the next multi word String
hasNextInt()	checks to see if there are more ints
hasNextDouble()	checks to see if there are more doubles
hasNext()	checks to see if there are more Strings

```
import java.util.Scanner;
```


The Scanner methods listed above include some of the most frequently used input methods and some methods that are used with loops to process multiple values from a input source.

Reading in ints



`keyboard` is a Scanner reference. `keyboard` stores the location / memory address of a Scanner.

The . dot is used to gain access to the Scanner Object.

`nextInt()` is the method being called on the Scanner reference `keyboard`.

Reading in Strings

```
out.print("Enter a string :: ");
String word = keyboard.next();
out.println(word);
```

INPUT
I love java.

OUTPUT
Enter a string :: I love java.
I

The `next()` method will read in the next text value entered. A numeric or non-numeric text value will be accepted.

In the example, the next text entered on the keyboard would be read in and placed in variable word.

The `next()` method would read up to the first whitespace encountered. Whitespace is any space, tab, or enter key.

Reading in Lines

```
out.print("Enter a line :: ");
String line = keyboard.nextLine();
out.println(line);
```

INPUT

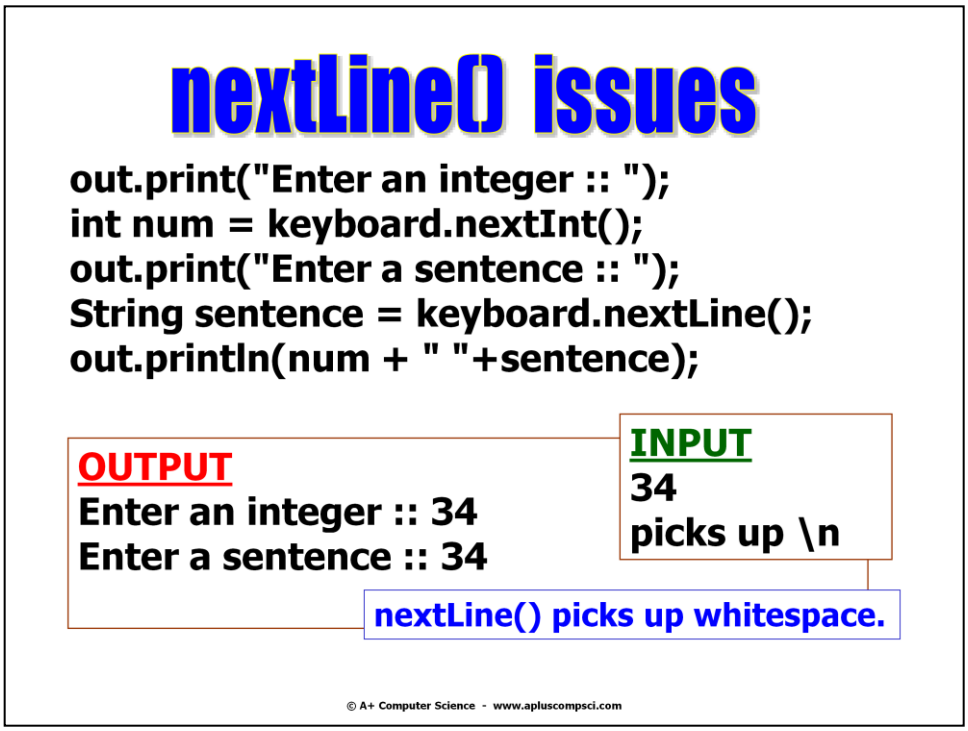
```
I love java.
```

OUTPUT

```
Enter a line :: I love java.
I love java.
```


The `nextLine()` method will read in an entire line of text including whitespace(enter keys, spaces, tabs, etc.). Any text value entered will be accepted, including a line containing spaces.

In the example, the next line of data entered on the keyboard would be read in and placed in variable sentence.



The `nextLine()` method will read in an entire line of text including the enter key. Any text value entered will be accepted, including a line containing spaces.

After 34 is typed in, the enter key must be pressed to get the system to register the 34.

`nextInt()` reads in the 34 and stores it in num.
`nextInt()` reads up to the enter key(\n) typed in after the 34.

`nextLine()` reads in the enter(\n) and stores it in sentence.

This is a problem.

nextLine() issues

```
out.print("Enter an integer :: ");
int num = keyboard.nextInt();
keyboard.nextLine();       //pick up whitespace
out.print("Enter a sentence :: ");
String sentence = keyboard.nextLine();
out.println(num + " "+sentence);
```

OUTPUT

```
Enter an integer :: 34
Enter a sentence :: picks up \n
34 picks up \n
```

INPUT

```
34
picks up \n
```

nextLine() picks up whitespace.

The `nextLine()` method will read in an entire line of text including the enter key. Any text value entered will be accepted, including a line containing spaces.

After 34 is typed in, the enter key must be pressed to get the system to register the 34.

`nextInt()` reads in the 34 and stores it in num. `nextInt()` reads up to the enter key(\n) typed in after the 34.

A `nextLine()` is placed after the `nextInt()` to read in the enter(\n). The additional `nextLine()` picks up the enter(\n) left behind by `nextInt();`

Now, `nextLine()` can read in the line and store it in sentence. The problem has been solved.

Multiple Inputs

INPUT
1 2 3 4 5

```
Scanner keyboard =
            new Scanner(System.in);

out.println(keyboard.nextInt());
out.println(keyboard.nextInt());
out.println(keyboard.nextInt());
```

OUTPUT

1
2
3

Scanner can be used to read in multiple values on one line as long as whitespace is entered in between each value on the line. If whitespace is not used to separate the values, the values would be considered one value.

For the example,if `1 2 3 4 5` is entered. Only values `1 2 3` are read in because the code only had 3 `nextInt()` method calls.

If `12345`, was entered with no spaces, then `12345` would be the first and only value read in.

scannerone.java

Scanner Class

```
Scanner keyboard = new Scanner(System.in);
```

object / reference

constructor

```
Scanner chopper = new Scanner("at  it  us");
```


The Scanner class contains several constructors.

One constructor takes in a reference to an `InputStream`. Typically, `System.in` is passed into this constructor.

A second constructor takes in a String reference as a parameter.

By overloading the constructors for Scanner, Java has provided more options and ways to use Scanner. By having multiple constructors, Scanner is much more dynamic and polymorphic.

Using Scanner for String Chopping

```
Scanner chopper =
        new Scanner("21   54   19");

out.println(chopper.nextInt());
out.println(chopper.nextInt());
out.println(chopper.nextInt());
```

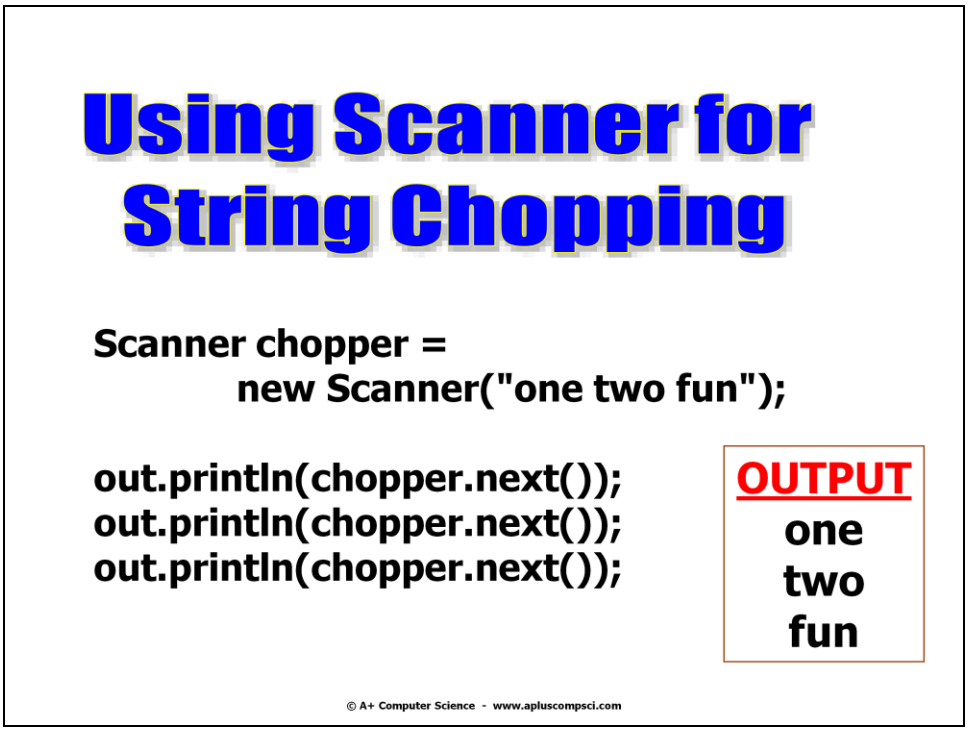
OUTPUT

21
54
19

`chopper` has been instantiated to refer to a Scanner Object that was constructed with a String. `chopper` will be used to chop up the String.

Each time `chopper.nextInt()` is called, the next integer in the list of integers is returned.

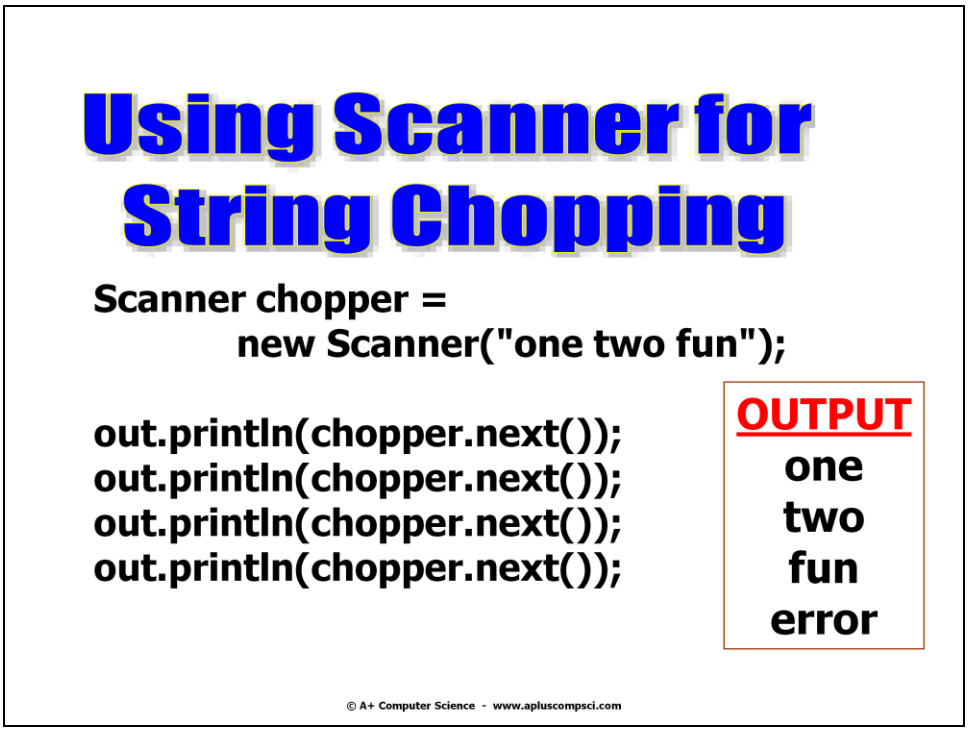
```
Scanner chopper;
chopper = new Scanner("4 9 6 1");
chopper.nextInt();     //returns 4
chopper.nextInt();     //returns 9
chopper.nextInt();     //returns 6
chopper.nextInt();     //returns 1
```



`chopper` has been instantiated to refer to a Scanner Object that was constructed with a String. `chopper` will be used to chop up the String.

Each time `chopper.next()` is called, the next one-word String in the list of Strings is returned.

```
Scanner chopper;
chopper = new Scanner("go it up wow");
chopper.next();          //returns go
chopper.next();          //returns it
chopper.next();          //returns up
chopper.next();          //returns wow
```



`chopper` has been instantiated to refer to a Scanner Object that was constructed with a String. `chopper` will be used to chop up the String.

Each time `chopper.next()` is called, the next one-word String in the list of Strings is returned.

```
Scanner chopper;
chopper = new Scanner("go it up");
chopper.next();          //returns go
chopper.next();          //returns it
chopper.next();          //returns up
chopper.next();          //exception thrown
```

scannertwo.java

Scanner methods used with loops

Scanner
frequently used methods

Name	Use
hasNextByte()	checks to see if there are more bytes
hasNextShort()	checks to see if there are more shorts
hasNextInt()	checks to see if there are more ints
hasNextLong()	checks to see if there are more longs
hasNextDouble()	checks to see if there are more doubles
hasNext()	checks to see if there are more Strings

All of these methods return true or false.

The Scanner methods listed above are used with loops to process values from a list.

While loop Review

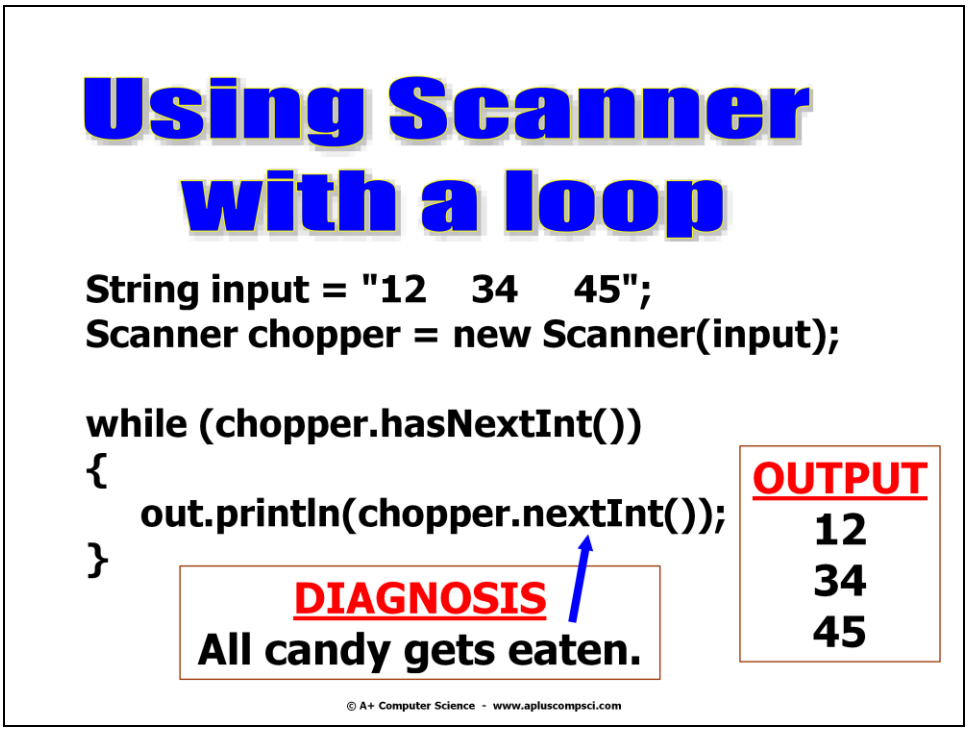
```
while ( I have candy )
{

}
```

DIAGNOSIS
Infinite Loop!
No candy
was eaten.

While loop Review

```
while ( I have candy )
{
  eat a piece of candy
}
```

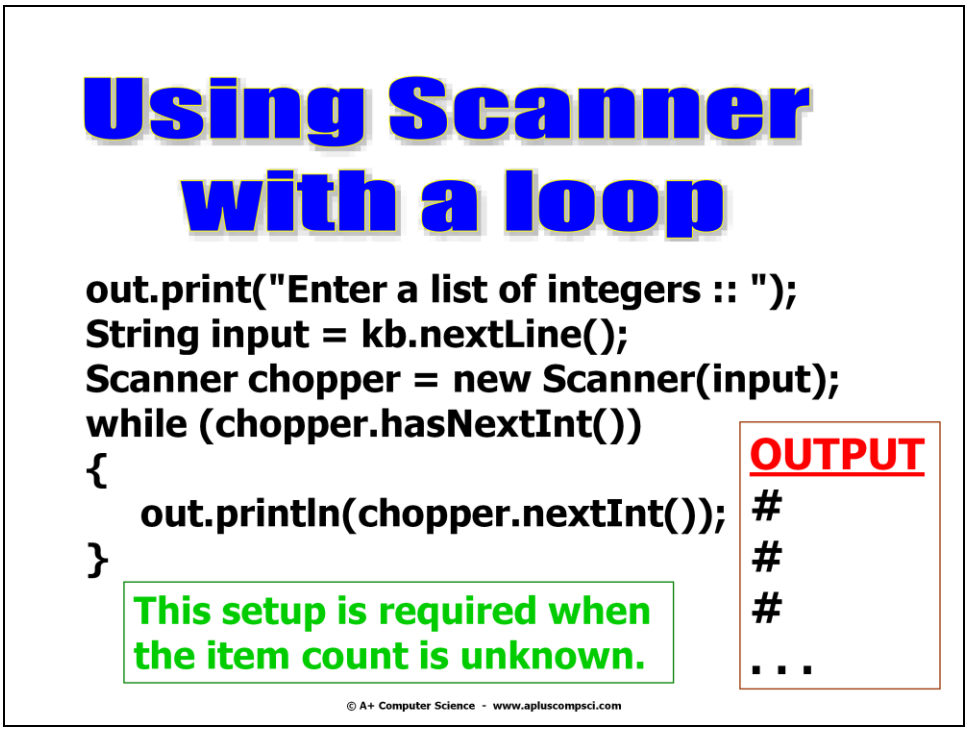
DIAGNOSIS
All candy gets eaten.



The while loop will iterate as long as the String being chopped has more int values left.
Each time an int is processed the chopper moves up to the next int.
The chopper.nextInt() is equivalent to eating a piece of candy from the prior example.

If no candy is eaten (chopper.nextInt()), the loop will run forever.

If the next value in the String is not an int, the while loop condition will fail as `chopper.hasNextInt()` will return false if encountering a non int value.



The while loop will iterate as long as the String being chopped has more int values left.

If the next value in the String is not an int, the while loop condition will fail as `chopper.hasNextInt()` will return false if encountering a non int value.

Input - `10 11 13 8 1 3 2 6`

For the input listed above, the while loop would iterate 8 times as there are 8 integers in the list.

Input - `10 11 13 bad 1 3 2 6`

For the input listed above, the while loop would iterate 3 times as the loop would fail when encountering `bad`.
`bad` is a String not an integer.

Using Scanner with a loop

```
out.print("Enter a sentence :: ");
String line = kb.nextLine();
Scanner chopper = new Scanner(line);
while (chopper.hasNext())
{
    out.println(chopper.next());
}
```

This setup is required when the item count is unknown.

The while loop will iterate as long as the String being chopped has more String values left. A String would be any combination of letters, numbers, and/or symbols. Any text value could be part of a String.

Input - `10 it 13.1 A 1 0.11 22 6ae y`

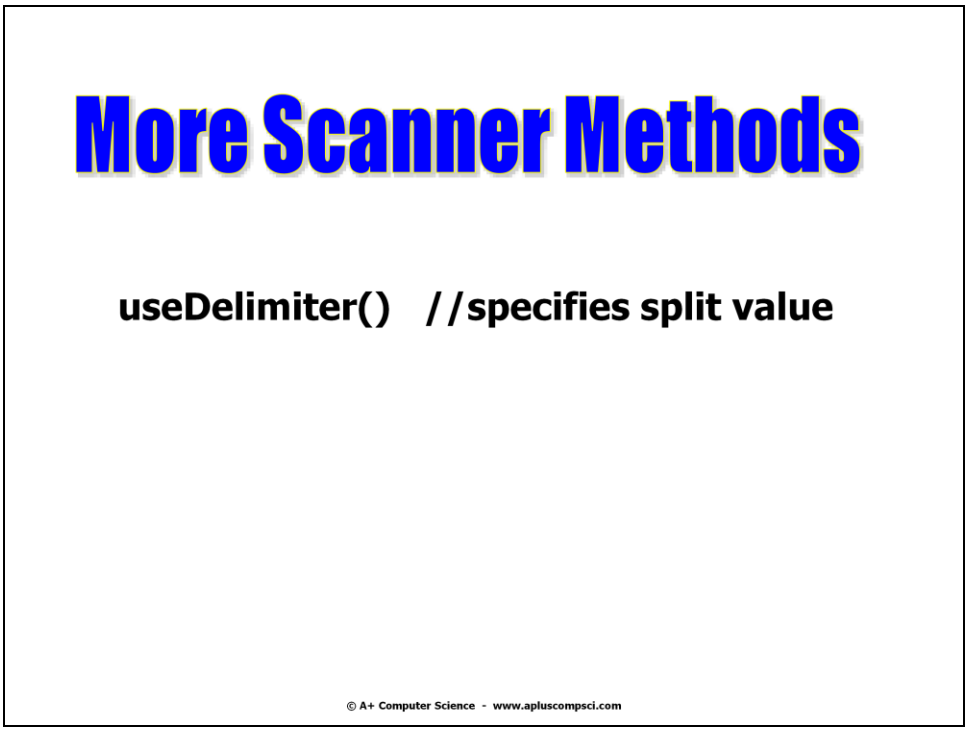
For the input listed above, the while loop would iterate 9 times as there are 9 String values in the list.

Input - `10 1.2 13 bad 1a 3226`

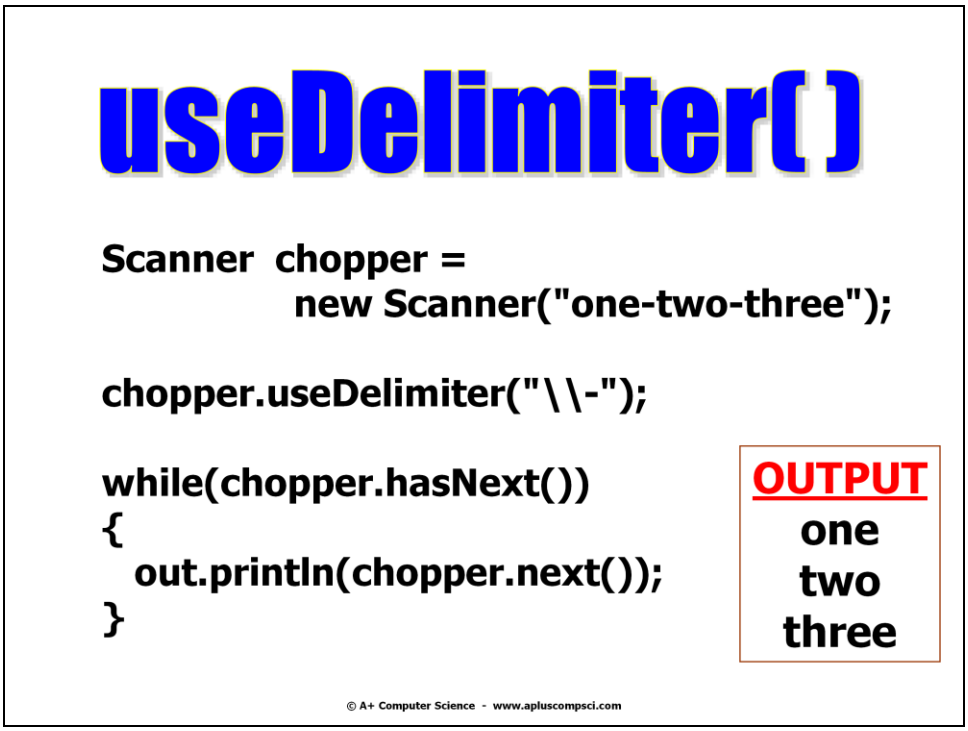
For the input listed above, the while loop would iterate 6 times as there are 6 String values in the list.

scannerthree.java

scannerfour.java



The `useDelimter()` method is used to tell the Scanner what value will be used to separate items. Whitespace is the default delimiter value. If a value other than whitespace is needed, a call to `useDelimter()` will be needed.



The `useDelimter()` method is used to tell the Scanner what value will be used to separate items. Whitespace is the default delimiter value. If a value other than whitespace is needed, a call to `useDelimter()` will be needed.

In the example above, - is being used as the delimiter. The `\\` is used out of habit. Regular expressions require `\\` to preface most punctuation that is typically used as special regular expression symbols if the symbol should be treated literally. It is a good habit to use `\\` on any punctuation that should be treated literally.

Once the new delimiter has been established, Scanner returns everything up to the first - when the first `chopper.next()` call is made. Then, Scanner returns everything after the first - and up to the second - is returned by the second call to `chopper.next()`. This process continues until all values have been processed.

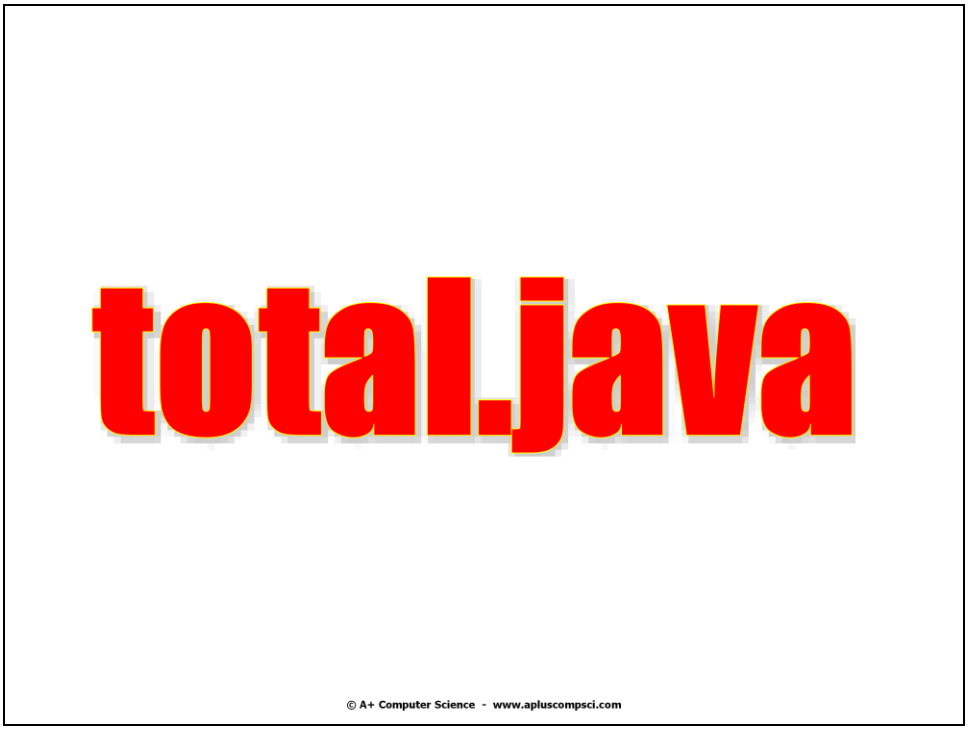
usedelimiter.java

Totaling Numbers With Loops

Using Loops To Total

```
Scanner keyboard = new Scanner(System.in);

out.print("How many numbers ::");
int count = keyboard.nextInt();
int sum = 0;

for(int i=0;i<count;i++) {
   out.print("Enter number " + (i+1) + " :: ");
   sum=sum+keyboard.nextInt();
}
out.println("total == " + sum);
```


The loop above is used to sum a group of values entered. The loop runs a set number of times and with each iteration, a value is entered. Each value entered is added to sum.


total.java
© A+ Computer Science - www.apluscompsci.com


Start work
on the labs
© A+ Computer Science - www.apluscompsci.com