Ternary Operator

A shortcut to assigning one of two values to a variable depending on a given condition

Like an **if-then-else** statement

Question mark comes after the condition

After the question mark, two values that can return are separated by a colon (:)

Takes 3 operands:	condition **?**	operand1 **:**	operand2
	Condition we're testing against	**First value** to assign if first condition was true	**Second value** to assign if first condition was false

Example:

```
int age = 20
```

```
boolean isOver18 =  (age == 20) ?  true :  false
```

is age equal to 20?	if it is, `isOver18= true`	if false, `isOver18 = false`

Operator Abbreviation

Original	Abbreviated
`result = result + 1;`	`result++;`
`result = result - 1;`	`result--;`
`result = result + 2;`	`result += 2;`
`result = result * 10;`	`result *= 10;`
`result = result / 3;`	`result /= 3;`
`result = result - 2;`	`result -= 2;`

Access Modifiers

These are java keywords

Allows defining the scope, how other parts of the code can access this code

Access Modifiers	Access Levels
public	Same Class, same package, other subclass, other package
protected	Same Class, same package, other subclass
no access modifier	Same Class, same package
private	Same Class

Code comments

These are used to describe methods for quick reference with an IDE

Start comment block:	`/**`	End comment block: `*/`
Describe method:	`Computes sum of two arguments`	
Describe parameters	`@param a`	an int operand
	`@param b`	an int operand
Describe what method returns:	`@ return`	the sum of a and b
Method described:	`public static int sum(int a, int b)`	

Full example:

```
/**
 * Computes the sum of two arguments.
  *
 * @param a an int operand to be added
 * @param b another int operand
  * @return the sum of a and b
 */
public static int sum(int a, int b)
```

For loop

For loops are used when you know exactly how many times you want to loop through a block of code

```
for (statement 1; statement 2; statement 3) { }
for (int i = 0; i < 5; i++) { }
```

sets a variable before the loop starts	Defines the condition for the loop to run. If true, the loop will start over again, if false, the loop will end.	increases a value (i++) each time the code block in the loop has been executed.

For-Each Loop

Used exclusively to loop through elements in an array:

```
for (type variab leName : arrayName) []
```

e.g.

```
String[] cars = {"Vo lvo ", " BMW ", " For d", " M az da"};
for ` (String i : cars)
System.ou t.p rin tln(i);
```

While Loops

A while loop will execute the enclosed statement as long as a boolean condition remains true.

Syntax: `while (boole an_ con dition) statemen`

The condition must be boolean.

If the condition never becomes false, the loop never exits, and the program never stops.

Example:

```
n = 1;
while (n < 4) {
   System.ou t.p rin tln(n + " squared is " + (n * n));
      n = n + 1;
}
```

Result:

```
1 squared is 1
2 squared is 4
3 squared is 9
```

When do you use each loop

for loop	if you know ahead of time how many times you want to go through the loop.
while loop	in almost all other cases.
do-while loop	if you must go through the loop at least once before it makes sense to do the test.

ArrayList

Create arraylist:

```
ArrayL ist <Ty pe> varName = new ArrayL ist <
```

e.g.

```
ArrayL ist <St rin g> names = new ArrayL ist <
```

Add object to arraylist	`varNam e.a dd( object)`
e.g.	`names.a dd ("Al ice ")`
Get size of arraylist:	`varNam e.s ize();`
e.g.	`names.s ize();`
Change object with index	`varNam e.s et( index, object);`
e.g.	`names.s et(0, " Ann a");`
Remove object with index	`varNam e.r emo ve( index)`
e.g.	`names.r em ove(0)`
Passing as a parameter example:	`public static double averag e(A e r> x)`

Arrays

Create array:	type[] varName = new type[size];
e.g.	double[] arr= new double[5]; //5 objects
Get length of array:	varName.length
e.g.	arr.length
Access object with index	varName[index]
	arr[1]

Reading input

Structure	
Import Scanner class:	`import java.u til.Sc anner;`
Create a scanner, assign it to a variable:	`Scanner scanner = new Scanne r(S yst em.i n);`
	`new Scanne r(..)` creates a new one
	System.in says scanner is to take input from the keyboard
Request user to input number	`System.ou t.p rin t("P lease input data: ");`
Read in number:	`myNumber = scanne r.n ext Int();`
Read in String	String myString = scanner.nextLine();
Read in double	String myDouble = scanner.nextDouble();
Read in char:	char myChar = scanner.next().charAt(0);

break statement

Inside any loop, the break statement will immediately get you out of the loop.

If you are in nested loops, break gets you out of the innermost loop

It doesn't make any sense to break out of a loop unconditionally; you should do it only as the result of an if test

break should not be the normal way to leave a loop

Use it when necessary, but don't overuse it.

Example:

```
 for (int i = 1; i <= 12; i++) {
if (badEg g(i))
 break;
}
```

Inheritance

subclass (child)	the class that inherits from another class
superclass (parent)	the class being inherited from
To inherit from a class we use	`extends`
class A extends B	means class A (subclass) inherits attributes and methods from class B(superclass)

Polymorphism