Break / Continue

(you’re not going anywhere, these are programming statements)

Break

When you want to exit a switch statement or a loop, you can use break.

```
while (true) {
    // some code here

    if (option.equals("exit")) {
        break;
    }
}

for (int i = 0; i < 10; i++)
{
    if (something) {
        break;
    }
}
```

Continue

If you want to jump to the next iteration in a loop, you can use continue.

```
while (true) {
    // some code here

    if (option.equals("menu")) {
        continue;
    }
}

for (int i = 0; i < s.length(); i++)
{
    if (s.charAt(i) == ' ') {
        continue;
    }
}
```

Switch

The switch statement is a cleaner way to represent several if statements/branches in your program.

Switch Statements

```
int r = rand.nextInt(3);
String choice = "";

switch (r) {
    case 0:
        choice = "rock";
        break;

    case 1:
        choice = "paper";
        break;

    case 2:
        choice = "scissors";
        break;
}
```

Switch Statements

You can have multiple lines of code inside a case option:

```
String name = "";

switch (courseNumber) {
    case 121:
        name = "Computer Programming I";
        totalCredits = totalCredits + 4;
        break;

    case 122:
        name = "Computer Programming II";
        totalCredits = totalCredits + 4;
        break;
}
```

Switch Statements

You can combine case statements together for similar options.

```
switch (courseNumber) {
    case 113:
    case 121:
    case 122:
        days = "TR";
        totalCredits = totalCredits + 4;
        break;

    case 271:
        days = "M";
        totalCredits = totalCredits + 2;
        break;
}
```

Switch Statements

For handling unexpected input or whatever is left, you can use default.

```
switch (r) {
    case 0:
        house = "Gryffindor";
        break;

    case 1:
        house = "Ravenclaw";
        break;

    // ... all the other choices ...

    default:
        house = "ERROR";
        break;
}
```

Enumerations (enum)

Enumerations (enum)

You may have a variable that has a limited amount of values or states. An enum can be used to avoid magic numbers and clarify your code.

Defining an enum is like defining your own data type.

```
// Define states for your Pokemon
public enum State {IDLE, DEFENDING, ATTACKING, SLEEPING}
```

Enumerations (enum)

You can see here that it's confusing what these numbers mean. If you use these numbers all over your code, you could make a mistake.

```
int pokemonState = 0;

switch (pokemonState) {

    case 0:
        // Do some stuff
        break;

    case 1:
        // Do some stuff
        break;
}
```

Enumerations (enum)

Here you can see it's much more clear (and easier to follow logically):

```
public enum State {IDLE, DEFENDING, ATTACKING, SLEEPING}

State pokemonState = State.IDLE;

switch (pokemonState) {

    case IDLE:
        // Do some stuff
        break;

    case DEFENDING:
        // Do some stuff
        break;
}
```

Let’s Code

Don’t Forget!

Check the syllabus / schedule for reading assignments and due dates!

Conditional Expressions

Conditional Expressions

It is really common to have an if statement with just two options. You can use a conditional expression as a shorthand for setting variables. The basic form is:

```
x = (condition) ? exprWhenTrue : exprWhenFalse
```

```
// Previous way
if (x == 0)
{
    name = "heads";
} else {
    name = "tails";
}

// Conditional Expression: Just 1 line
name = (x == 0) ? "heads" : "tails";
```

Conditional Expressions

Here is another example:

```
int grade = scan.nextInt();

// Previous way
if (grade >= 65) {
    result = "Pass";
} else {
    result = "Fail";
}


// Conditional Expression: Just 1 line
result = (grade >= 65) ? "Pass" : "Fail";
```

do - while

When you want to do something at least once.

do while

With a while loop, you only enter the loop if the condition is true.

```
while (quit == false) {
    // do something
}
```

With a do - while loop, you always run the loop at least once, and evaluate the condition at the end.

```
do {
    // do something
} while (quit == false);
```

do while

This is handy for programs with menus or input validation.

```
String name = "";

do {
    System.out.println("Enter your full name: ");
    name = scan.nextLine();
} while (name.length() <= 0);
```

Scope

(nothing to do with your breath, it's a programming concept)

Scope

The area where a variable is available is referred to as its scope.

```
String name = "";

while (quit == false) {
    name = scan.nextLine();
    int x = 0;
}
System.out.println(name);     // This is ok!
System.out.println(x);        // Will not compile!

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

System.out.println(i);        // Will not compile!
```

File Input / Output

(this is a complicated topic, but this is enough to get started)

File Input / Output

The Scanner class that we used for reading keyboard input (System.in) can also be used to read from strings or files.

```
// Scanner - String example

String lyric = "You used to call me on my cell phone.";
Scanner scan = new Scanner(lyric);

String firstWord = scan.next();
```

```java
import java.util.Scanner;

import java.io.FileInputStream;
import java.io.IOException;

public class FileInputExample {

    public static void main(String[] args) throws IOException
    {
        FileInputStream infs = new FileInputStream("test.txt");

        Scanner inputScan = new Scanner(infs);

        while (inputScan.hasNextLine())
        {
            String line = inputScan.nextLine();
            System.out.println(line);
        }

        infs.close();
    }
}
```

```java
import java.util.Scanner;

import java.io.IOException;
import java.io.FileOutputStream;
import java.io.PrintWriter;

public class FileOutputExample
{
    public static void main(String[] args) throws IOException
    {
        Scanner scan = new Scanner(System.in);
        System.out.println("Add a line: ");
        String line = scan.nextLine();


        FileOutputStream outfs = new FileOutputStream("test.txt", true);
        PrintWriter pw = new PrintWriter(outfs);

        pw.println(line);
        pw.flush();

        outfs.close();
    }
}
```

Let’s Code

Don’t Forget!

Check the syllabus / schedule for reading assignments and due dates!