Comparing Strings

- If the == operator is used
  - Java compares the addresses where the String objects are stored, not the letters in the String
  - For example:
    > String a = "hi";
    > String b = "hi";
    > a == b
      false

- Use the String class' equals method to compare two Strings for equality
  > a.equals(b)
  true
  > b.equalsIgnoreCase("HI")
  true

String class is part of Java Language, just like Math class

Packages and import Statements

- If a class is not part of java language i.e. java.lang, you'll see package name
- What is a package?
  - Basically it's a directory that has a collection of related classes
  - E.g. Random Class description contains: java.util.Random
  - Indicating that the Random code is stored in java/util/Random.class somewhere on your machine.
  - The java/util directory/folder is known as the "util", or utility package.
- Since Random is not part of Java Language we need to tell Java where to find it by saying
  - import java.util.Random;
  - Another way is to use the asterisk "wildcard character": import java.util.*;

Random Class

- A class to create Random numbers
- Constructor Summary shows the objects of this type can be created
  - E.g. Random r = new Random();
- Method Summary shows that it can generate random values of types:
  - integers, doubles etc.
  - E.g. r.nextInt(6) – Generate a integer numbers between 0 (inclusive) and 6 (exclusive)
  - How do I generate a number between 1 and 6?
Number “width”

- Numeric types are considered wider or narrower than other numeric types.
  - This is based partly on how much memory space they occupy.
  - Also based on how large a number it can hold.
- Java doesn’t mind if you assign a narrow value to a wide variable: `int n = 3;`
- Java is not happy if you assign a wide value to a narrow variable: `int n = 3.5; //illegal`
- But if you want to narrow (assign a wider type to a narrower type), you have to cast it:
  - `double d = 3.5;`  
  - `i = (int) d; //legal due to casting`

Casts

- You can convert (cast) one numeric type to another.
- When you widen, no explicit cast is necessary.
  - E.g. `double d = 5;`
  - But it doesn’t hurt.
- When you narrow, an explicit cast is required.
  - This requirement is made to help avoid accidental loss of precision.
- Casting tells Java that the value in the wider type will fit in the narrower type.
- Java checks to make sure that the cast works, and gives you an error if it didn’t.

char

- The primitive type `char`
  - Just stored as numbers.
  - Each char as a unique integer value (based on Unicode standard).
- You can use characters in arithmetic (they will automatically be converted to `int`).
  - `char ch = 'A';`
  - `ch + 1`  
  - `66`
  - `char ch2 = (char) (ch + 1) // cast result back to char`
  - `B`

The `do-while` statement

```
do {
  statement(s)
} while (condition);
```

1. Do the statement(s) at least once.
2. If the `condition` is `true`, re-execute `statement(s)`; repeat step 2.
   false: we’re done with the loop.
Example #1: do-while

```java
//Example 1
int count = 1;
do {
    System.out.println("Count is: " + count);
    count++;
} while (count <= 11);
//Example 2
char input;
do {
    input = getInputFromUser();
    processInput(input);
} while (input != 'q');
```

A for/while-loop with continue

```java
for (expr1; condition1; expr2){
    ...
    if (condition2){
        continue; // evaluate expr2, then condition1
    }
    ...
}
while (condition1){
    ...
    if (condition2){
        continue; // go up and re-evaluate condition1
    }
    ...
}
```

Note: Continue for do-while is same as while loop

The increment operator

- `++` adds 1 to a variable
  - It can be used as a statement by itself, or within an expression
  - It can be put before or after a variable
  - If before a variable (pre-increment), it means to add one to the variable, then use the result
  - If put after a variable (post-increment), it means to use the current value of the variable, then add one to the variable

Examples of `++`

```java
int a = 5;
a++;
// a is now 6
int e = 5;
int f = e++;
// e is 6, f is 5
int b = 5;
++b;
// b is now 6
int x = 10;
int y = 100;
int z = ++x + y++;
// x is 11, y is 101, z is 111
int c = 5;
int d = ++c;
// c is 6, d is 6
```

Confusing code is bad code, so this is very poor style
The decrement operator

- The decrement operator "--" subtracts 1 from a variable.
- Used similarly as ++ operator.

```java
int a = 5;
> a--; // a is now 4
> --a; // a is now 3
```

Confusing code is bad code, so this is very poor style.

Syntax of the switch statement

- The syntax is:
  ```java
  switch (expression) {
    case value1 : statements ; break ;
    case value2 : statements ; break ;
    ...(more cases)...
    default : statements ; break ;
  }
  ```

- The expression must yield an integer or a character.
- Each value must be a literal integer or character.
- Notice that colons (:) are used as well as semicolons.
- The last statement in every case should be a break;
- I even like to do this in the last case.
- The default: case handles every value not otherwise handled.

Example switch statement

```java
switch (cardValue) {
  case 1:
    System.out.print("Ace");
    break;
  case 11:
    System.out.print("Jack");
    break;
  case 12:
    System.out.print("Queen");
    break;
  case 13:
    System.out.print("King");
    break;
  default:
    System.out.print(cardValue);
    break;
}
```