Loops

- Three loops
  - While
  - Do-while
  - For
  - Break, Continue statements
Why Loops?

- A lot of repetitions in programs
  - Tedious to repeat to write the same statements
- When a repeating pattern exists, good to use a Loop

Examples

- Writing the following statement a hundred times
  - System.out.println("Welcome to Java!");
- Displaying numbers from zero to one hundred
  - 0, 1, 2, 3, ..., 100
- Repeat the rock-paper-scissor game if the end-user wants to play more
A while loop provides iteration processing.

- Iteration: executing a code segment more than once
WHILE Loop

while (Boolean_expression)
{
    statement(s)
}

WHILE Loop

- Display “Hello World” one hundred times

```java
int i = 1; //initialize count
while(i <= 100)
{
    System.out.println("Hello World");
    i++; //increase count by 1
}
```
WHILE Loop

- Display all numbers between 1 and 100
- Display all even numbers between 1 and 100

```java
int i = 1;
while (i <= 100)
{
    if (! (i % 2))
        System.out.println(i);
    i++;
}
```
WHILE Loop

- Display ASCII characters

Decimal, Hexadecimal, Octal, and Chr

```java
int c = (int)'A';
System.out.printf("%d, %X, %o, %c", c, c, c, c);
```
Guess Numbers

Write a program that randomly generates an integer between 1 and 10 (inclusive) and prompts the user to enter a number continuously until the number matches the random number.
WHILE Loop

- Display a program menu
WHILE Loop

- Display a program menu

Choose a menu (enter a number)
1: Save Game
2: Load Game
3: Options
4: Help
5: Resume Game
6: Exit
Nested Loops

- Inner loop that appears in the loop body of another loop.
  - Draw a grid (3*10)
    
    * * * * * * * * * *
    * * * * * * * * * *
    * * * * * * * * *
    * * * * * * * * *

  - multiplication table
    
    2 * 1 = 2, 2 * 2 = 4, ...
    3 * 1 = 3, 3 * 2 = 6, ...
Nested Loops

- **Draw a grid (3*10)**

```java
int i = 0;    // index for row
int j = 0;    // index for column
while (i < 3)
{
    while (j < 10)
    {
        System.out.print("* ");
        j++;
    }
    System.out.println("\n");
    j = 0;
    i++;
}
```
Caution

- Don’t use floating-point values for equality checking in a loop control. Since floating-point values are approximations for some values, using them could result in imprecise counter values and inaccurate results. Consider the following code for computing $1 + 0.9 + 0.8 + \ldots + 0.1$:

```java
double item = 1; double sum = 0;
while (item != 0) { // No guarantee item will be 0
    sum += item;
    item -= 0.1;
}
System.out.println(sum);
```
Break statement

- Terminates the execution of the nearest enclosing WHILE, FOR, DO-WHILE, or SWITCH statements in which it appears

```java
while (expression)
{
    statement
    break;
}
```
int i = 0;
while (true)
{
    System.out.print(i);
    i++;
    break;
}
Break statement

```java
int i = 0;
while (true)
{
    System.out.print(i);
    i++;
    i++;
    if (i > 10)
    {
        break;
    }
}
```
Continue statement

- Related to break, but less often used
- It causes the next iteration of the enclosing for, WHILE, or DO-WHILE loops to begin
- In the WHILE and DO-WHILE, this means that the condition part is executed immediately,
int i = 0;
while (i < 50)
{
    if (i % 2 != 0)  // skip odd numbers
    {
        i++;  
        continue;
    }
    // do something for even numbers
    System.out.printf("%d\t", i);
    i++;
}

Continue statement