Loops provide substantial computational power.
Loops bring an efficient way of programming.
Loops could consume a lot of time.\footnote{We will introduce the analysis of algorithms soon.}
while Loops

A while loop executes statements repeatedly while the condition is true.

```java
... while (condition) {
    // loop body
}
...```

- The condition should be a boolean expression which determines whether or not the execution of the body occurs.
- If true, the loop body is executed and check the condition again.
- Otherwise, the entire loop terminates.
```
count = 0;

(count < 100)?
true
System.out.println("Welcome to Java!");
count++;
false

loop-continuation-condition?
false

Statement(s) (loop body)
false

true
```
Example

Write a program which sums up all integers from 1 to 100.

- In math, the question can be written as:
  \[ \text{sum} = 1 + 2 + \cdots + 100. \]

- But this form is not doable in the machine.²

²We need to develop computational thinking. Read http://rsta.royalsocietypublishing.org/content/366/1881/3717.full or http://blog.orangeapple.tw/posts/what-is-computational-thinking/.
• Normally, the machine executes the instructions *sequentially*.
• So one needs to decompose the math equation into several steps, like:

```java
...  
    int sum = 0;
    sum = sum + 1;
    sum = sum + 2;
    .
    .
    .
    sum = sum + 100;
...  
```

• It is obvious that many similar statements can be found.
• Using a `while` loop, the program can be rearranged as follows:

```java
... 
int sum = 0;
int i = 1;
while (i <= 100) {
    sum = sum + i;
    ++i;
}
... 
```

• You should guarantee that the loop will terminate as expected.
• In practice, the number of loop steps (iterations) is unknown until the input data is given.
Malfunctioned Loops

- It is really easy to make an infinite loop.

```java
...  
while (true);  
...
```

- The common errors of the loops are:
  - never start
  - never stop
  - not complete
  - exceed the expected number of iterations
Write a program which asks the sum of two random integers and lets the user repeatedly enter a new answer until correct.

```java
Scanner input = new Scanner(System.in);
int x = (int) (Math.random() * 10);
int y = (int) (Math.random() * 10);
int ans = x + y;

System.out.println(x + " + " + y + " = ? ");
int z = input.nextInt();

while (z != ans) {
    System.out.println("Try again? ");
    z = input.nextInt();
}
System.out.println("Correct.");
input.close();
...
Loop Design Strategy

• Writing a correct loop is not an easy task for novice programmers.
• Consider 3 steps when writing a loop:
  • Find the pattern: identify the statements that need to be repeated.
  • Wrap by loops: put these statements in the loop.
  • Set the continuation condition: translate the criterion from the real world problem into computational conditions.\(^3\)

\(^3\)Not unique.
Another common technique for controlling a loop is to designate a special value when reading and processing a set of values.

- This special input value, known as a sentinel value, signifies the end of the loop.
- For example, the operating systems and the GUI apps.
Write a program which sums over positive integers from consecutive inputs and then outputs the sum when the input is nonpositive.

```java
... 
int total = 0, price = 0;
Scanner input = new Scanner(System.in);

System.out.println("Enter price?");
price = input.nextInt();
while (price > 0) {
    total += price;
    System.out.println("Enter price?");
    price = input.nextInt();
    // These two lines above repeat Line 5 and 6?!
}

System.out.println("Total = " + total);
input.close();
... 
```
do-while Loops

A **do-while** loop is similar to a while loop except that it **does** execute the loop body first and then checks the loop continuation condition.

```java
... 
  do {
    // loop body
  } while (condition); // Do not miss the semicolon!
... 
```

- Note that there is a semicolon at the end of the **do-while** loop.
- The **do-while** loops are also called **posttest** loops, in contrast to while loops, which are **pretest** loops.
Write a program which sums over positive integers from consecutive inputs and then outputs the sum when the input is nonpositive.

```java
... int total = 0, price = 0;
Scanner input = new Scanner(System.in);

do {
    total += price;
    System.out.println("Enter price?");
    price = input.nextInt();
} while (price > 0);

System.out.println("Total = " + total);
input.close();
...
for Loops

A for loop generally uses a variable to control how many times the loop body is executed.

```java
... for (init_action; condition; increment) {
    // loop body
}
...```

- **init-action**: declare and initialize a variable
- **condition**: set a criterion for loop continuation
- **increment**: how the variable changes after each iteration
- Note that these terms are separated by semicolons.
Example

Sum from 1 to 100

Write a program which sums from 1 to 100.

```java
... int sum = 0;
for (int i = 1; i <= 100; ++i) {
    sum = sum + i;
}
...
```

- Compared to the `while` version,

```java
... int sum = 0;
int i = 1;
while (i <= 100) {
    sum = sum + i;
    ++i;
}
... 
```
Example: Selection Resided in Loop

Display all even numbers

Write a program which displays all even numbers smaller than 100.

- An even number is an integer of the form $x = 2k$, where $k$ is an integer.
You may use the modular operator (%).

```java
... 
for (int i = 1; i <= 100; i++) {
    if (i % 2 == 0) System.out.println(i);
}
...
```

Also consider this alternative:

```java
... 
for (int i = 2; i <= 100; i += 2) {
    System.out.println(i);
}
...
```

How about odd numbers?
Numerical Example: Monte Carlo Simulation

- Let $m$ be the number of sample points falling in the region of the quarter circle shown in the next page, $n$ be the total number of sample points.
  - Simply use `Math.random()` to generate a value between 0 and 1 (exclusive).
- Write a program which estimates $\pi$ by
  $$\hat{\pi} = 4 \times \frac{m}{n}.$$
- Note that $\hat{\pi} \to \pi$ as $n \to \infty$ by the law of large numbers (LLN).

---

4See https://en.wikipedia.org/wiki/Monte_Carlo_method.
Numerical Example: Bisection Method for Root-Finding

• Assume that \( f(x) = x^3 - x - 2 \).
• Consider to find a root between \([a, b] = [1, 2]\) as initial guess.\(^5\)
• Write a program which calculates the approximate root \( \hat{r} \) under this requirement by using the bisection method.
  • In particular, you may set an error tolerance, say \( \epsilon = 1e^{-9} \), to strike a balance between efficiency and accuracy.

\(^5\)For most of numerical algorithms, say Newton’s method, an initial guess is a must. Even more, the solution is severely sensitive to the initial guess for some cases.