While and If

- Branching – use if/else
- Loops – use while
- In theory, you could use only these two control structures for your programs.
- However, it is often convenient to use other structures as well.
while

while (booleanExpr) {
  // statements to repeat while booleanExpr is true
}

A while loop tests the continuation condition first, then executes the statements in the body.
**do..while**

Sometimes it is more convenient continuation condition at the end of the loop, instead of the beginning.


do {
    // body of loop
} while(booleanExpr);

while vs do..while

Having two types of loops doesn’t actually make the language more powerful. We can write one in terms of the other.

do..while rewritten as while

```java
do {
    loopBody
} while (textExpr);
```

while rewritten as do..while

```java
while (testExpr) {
    loopBody
}
```

if(testExpr) {
    do {
        loopBody
    } while (testExpr);
}
break out of loops

Sometimes you want to break out of a middle of a loop.

```java
Scanner sc = new Scanner(System.in);
int n; // holds user number
while(true) {  // oh no, an infinite loop!
    System.out.println("Enter an integer");
    if(sc.hasNextInt()) {
        // user actually entered a number. Yay!
        n = sc.nextInt();
        // Use break to jump out of the loop
        break;
    } else {
        // Give user another chance to get it right
        System.out.println(sc.next() + " isn’t an integer!"
    }
}
// program continues here after break
```
for loops

Many while loops have the general form:

```
init
while(testExpr) {
    body
    update
}
```

We can use a for loop to combine initialization, continuation testing, and updating in the first line.

```
for(init; testExpr; update) {
    body
}
```

We don’t get any new power by adding for to the language, but for some kinds of problems, it can be easier to use.