Student objects:

Student

Student

Student

Student

StudentList references Student objects:

(0) (1) (2) (3)
**while statement**

- **iteration**: a process in which an operation is performed several times.
- **Syntax:**
  ```
  while ( condition )
  statement
  ```
**while statement (cont.)**

- The component statement is called the *body*.
- Executing the body should have the potential of changing the value of the condition.
- It is possible that the condition of a “while” statement will remain true no matter how many times the body is executed. This is an “infinite loop.”
while loops and arrays.

```java
while (more list elements to process )
    process the next element

e.g.
    int index;
    index = 0;
    while (index < array.length) {
        process array[index];
        index = index + 1;
    }

- We must guarantee that the while condition eventually will become false.
```
**while example - summing items of an array**

```java
public double finalAverage(Students[] studentList)
    The average (mean) of the final exam grades of the specified Students.
    require:
    studentList.length > 0
public int finalExam()
    This Student’s grade on the final exam.

public double finalAverage(Students[] studentList)
{
    int i, sum, count;
    count = studentList.length;
    sum = 0;
    i = 0;
    while (i < count) {
        sum = sum + studentList[i].finalExam();
        i = i + 1;
    }
    return (double)sum / (double)count;
}
```
while example illustrated
Summing selected elements of a List

- Consider what to do if some students did not take the final exam.
/**
 * The average (mean) of the final exam grades of the
 * specified Students. Negative exam grades are not included
 * in computation of the average.
 * require:
 * students.size() > 0
 * students.get(i).finalExam() >= 0
 * for some i, 0 <= i < students.size()
 */

public double average (StudentList students) {
    int i;    // index of students
    int sum;  // sum of non-negative grades up to, but not
               // including, the i-th Student
    int count;  // number of non-negative grades up
                 // to, but not including, the i-th
    int number; // number of Students

    number = students.size();
    sum = 0;
    count = 0;
    i = 0;

    while (i < number) {
        if (students.get(i).finalExam() >= 0) {
            sum = sum + scores.get(i).finalExam();
            count = count+1;
        }
        i = i+1;
    }

    return (double)sum / (double)count;
}
Finding the minimum

/**
 * The lowest final exam grades of the
 * specified Students.
 * require:
 * students.size() > 0
 */

public int minFinalExam (Students [ ] studentList){
    int i;
    int low;
    low = studentList[ 0 ].finalExam( );
    i=1;
    while ( i < studentList.length){
        if (studentList[ i ].finalExam( ) < low)
            low=studentList[ i ].finalExam( );
        i = i+1;
    }
}
Determining if an object is in a List

```java
public boolean contains (Students [] studentList, Students s) {
    int i;
    int size;
    size = studentList.length ;
    i=0;
    while ( i< studentList.length && studentList[i] != s )
        i = i+1;
    return i < size;
}
```
What does “equal” mean?

- Two reference values are equal if they refer to the same object.
More on “equal”

- Consider a Date class, that has components day, month, and year.
- If distinct Date objects are created to represent the same date, would these objects be “equal”? Must they refer to the same object?
Even more on “equal”

To determine if they are “equal” dates, we must compare their day, month, and year.

```java
public boolean equals (Object obj){
    Require.condition(obj instanceof Date);
    Date d = (Date) obj;
    return this.year()==d.year() &&
           this.month()==d.month() &&
           this.day()==d.day();
}
```

// instanceof returns true if the object is
// an instance of
// the class
Two meanings of equality

*Identity or referential equality:*

- an object is only equal to itself;
- references are equal only if they contain the same reference value: that is, only if they reference the same object;
- the meaning of the equality operator “==”;
- the default meaning of the method *equals.*

*State equality:*

- two objects are equal if their states are equal;
- must be explicitly defined by the method *equals* in the class definition.
indexOf method

```java
public int indexOf (Student obj) {
    int i;
    int length;
    length = this.size();
    i = 0;
    while (i < length && !obj.equals(get(i)))
        i = i+1;
    if ( i < length)
        return i;
    else
        return -1; // item not found
}
```
Loop structure

**initialization**

**while** (condition) {
  body
}

**conclusion**
for statement

for ( initialization; condition; updateStatement )
statement
for statement example

```java
int i;

for ( i = 0; i < list.length; i = i+1) 
    process list[ i ];
```