Lecture 21: Switch, Break, Continue

Last time:
1. Continuation of Rational numbers
2. switch
3. break
4. Case continuation (or “fall through”)

Today:
1. break / continue
2. Intro to arrays
3. Copying arrays and making arrays bigger
4. Array lengths and out-of-bounds indexing
Why Use `switch`?

- `switch` can also be implemented using `if-else`.
- `switch` also restricted in terms of data types in control statements.
- Including `break` statements is a pain.
- However
  - `switch` often more efficient (compiler generates better code).
  - Code can be more compact because of case-continuation behavior.
  - Sometimes case analysis is clearer using `switch`.
More about break

- `break` can also be used to exit immediately from any loop
  - while
  - do-while
  - for
- e.g. “Read numbers from input until negative number encountered”
  ```java
  Scanner sc = new Scanner (System.in);
  int n;
  while (true) {
    n = sc.nextInt ();
    if (n < 0)
      break;
    else
      <process n>;
  }
  ```
- Loop only terminates when `break` executed
- This only happens when n < 0
Warning about `break`

- Undisciplined use of `break` can make loops impossible to understand
  - Termination of loops without `break` can be understood purely by looking `while`, `for` `parts`
  - When `break` included, arbitrary termination behavior can be introduced

- Rule of thumb: use `break` only when loop condition is always true (i.e. `break` is only way to terminate loop)
**continue Statement**

- `continue` can also be used to affect loops
  - `break` halts loops
  - `continue` jumps to bottom of loop body
- **Following prints even numbers between 0 and 10**
  ```java
  for (int i = 0; i <= 10; i++){
    if (i % 2 == 1)
      continue;
    System.out.println (i);
  }
  ```
  - **Effect of continue statement is to jump to bottom of loop immediately when i is odd**
  - This bypasses `println`!
- `continue` should be avoided
  - Confusing
  - Easy equivalents exist (e.g. `if-else`)
  - Included in Java mainly for historical reasons
Data Structures and Arrays

- **Data structures**: mechanisms for storing data in a structured way
- We have seen simple data structures implemented as classes:
  - `Date.java`
    - “Date” data stored as month / day / year
    - Auxiliary data (e.g. separator) also stored
  - `Rational.java`
    - Rational number data stored as numerator / denominator pair
- **Arrays** are a very useful data structure provided by Java and other programming languages
  - Array: sequence of variables of the same type
  - Individual elements of sequence can be uniformly referenced / updated / etc.
Creating Arrays

- Arrays are objects (hence allocated on heap)
- To create an array, use `new`
- Java has special syntax for array type / constructor:
  ```java
  int[] a = new int[5];
  ```
  - `int[]` indicates `a` is array of integer variables
  - `int[5]` indicates `a` will have 5 elements
- Like other objects, “instance variables” of array = cells in array are assigned default values (0 / null / etc.) when array created
Array Indexing

- Java provides a special syntax for uniformly accessing cells in an array
  - Assume previous declaration of `a`:
    ```java
    int[] a = new int[5];
    ```
  - This in effect creates five `int` variables “named”:
    ```java
    ```
  - To modify contents of cell #2 to 6:
    ```java
    a[2] = 6;
    ```
- This access mechanism is called array indexing
  - In Java / C / C++, array cells are indexed beginning at 0 and going up to n-1 (n is number of cells)
  - Beware of starting a 0!
Example

```java
int[] a = new int[5];
a[2] = 6;
```
Loops Are Useful for Processing Arrays

- Loop counter can be used to iterate through index values
- Each iteration of loop thus processes a different array element
- See examples
A Common Programming Idiom

- To process all elements in array `a`...
- Do following:
  ```
  for (int i = 0; i < a.length; i++){
      ...a[i]...
  }
  ```
- Use fresh loop counter to avoid overwriting another variable of same name elsewhere
- **Use** `i < a.length`, **not** `i <= a.length` (why?)
Copying Arrays

- Does the following copy a into b?
  ```java
  int[] a = new int[5];
  int[] b = null;
  b = a;
  No: a, b are aliases
  ```

- How to make a copy? For now, use loop:
  ```java
  int[] a = new int[5];
  int[] b = null;
  b = new int[a.length];
  for (int i = 0; i < a.length; i++)
    b[i] = a[i];
  ```
Making Arrays Bigger

- Suppose we want to make an array bigger by adding an element.
- Does the following work?
  ```java
  int[] a = new int[5];
  a.length++;
  ```
- No!
  - We get the following:
    ```java
    Exception in thread "main" java.lang.Error:
    Unresolved compilation problem:
    The final field array.length cannot be assigned
    at Sample.main(Sample.java:15)
    ```
  - `a.length` is immutable
  - No assignment is allowed
To Make an Array Bigger...

- Create a new larger array object
- Copy old array contents into new object
- Assign address of new object to variable
  ```java
  int[] a = new int[5];
  {
    int[] temp = new int[a.length + 1];
    for (int i = 0; i < a.length; i++)
      temp[i] = a[i];
    a = temp;
  }
  ```
  - New variable temp created to hold copy
  - New block created to ensure temp does not interfere with another variable of the same name
  - Previous contents of a become garbage
More on Array Indexing

- Recall: if \( a \) is an array of \( n \) elements, then
  - Initial element is \( a[0] \)
  - Last element is \( a[n-1] \)
- Each array as an instance variable, length, storing the number of elements in the array
  - Consider:
    ```java
    int[] a = new int[10];
    System.out.println(a.length);
    ```
  - 10 is printed
- What happens?
  ```java
  int[] a = new int[10];
a[10] = 0;
  ```
  - Error!
    ```java
    Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 10 at Sample.main(Sample.java:11)
    ```
  - The cells in \( a \) are \( a[0], \ldots, a[9] \)
  - \( a[10] \) is like an undefined variable
- This kind of error is called an index out-of-bounds error
  - If you write \( a[\text{exp}] \) ...
  - \( \ldots \) and \( \text{exp} \) evaluates to something \( \geq a.\text{length} \) ...
  - \( \ldots \) then Java reports an ArrayIndexOutOfBoundsException error