University at Buffalo
The State University of New York

cse@buffalo

CSE115 / CSE503
Introduction to Computer Science I

Dr. Carl Alphonce

343 Davis Hall

alphonce@buffalo.edu

Office hours:

Tuesday 10:00 AM – 12:00 PM*

Wednesday 4:00 PM – 5:00 PM

Friday 11:00 AM – 12:00 PM

OR request appointment via e-mail

**Tuesday adjustments: 11:00 AM – 1:00 PM on 10/11, 11/1 and 12/6*

ROADMAP

Last time

- Collections

Today

- Iterators
- Inheritance (maybe)

Coming up

- Inheritance

ITERATORS and the ITERATOR DESIGN PATTERN

Collection<E> has this method:

public Iterator<E> iterator()

returns an iterator over elements currently in collection

java.util.Iterator<E>

An iterator guarantees to visit all members of a collection exactly once between the time it is created to the time its hasNext() method returns false

java.util.Iterator<E> interface

Three methods; we focus on these two:

public boolean hasNext()

- returns true if there are items of the underlying collection that still need to be visited
- returns false if there are no unvisited items

public E next()

- returns one of the unvisited items, if any
- should not be called if hasNext() returns false

hasNext()'s return type is boolean

call to hasNext() can be used to control while loop

Example

```
java.util.Collection<String> coll;
coll = new java.util.LinkedList<String>();
coll.add("Fred");
coll.add("Wilma");
coll.add("Pebbles");
coll.add("Dino");
coll.add("Fred");
System.out.println("The collection: " + coll);
java.util.Iterator<String> it;
it = coll.iterator();
if (it.hasNext()) { System.out.println(it.next()); }
if (it.hasNext()) { System.out.println(it.next()); }
if (it.hasNext()) { System.out.println(it.next()); }
if (it.hasNext()) { System.out.println(it.next()); }
if (it.hasNext()) { System.out.println(it.next()); }
```

Notice repetition

```
java.util.Collection<String> coll;
coll = new java.util.LinkedList<String>();
coll.add("Fred");
coll.add("Wilma");
coll.add("Pebbles");
coll.add("Dino");
coll.add("Fred");
System.out.println("The collection: " + coll);
java.util.Iterator<String> it;
it = coll.iterator();
if (it.hasNext()) { System.out.println(it.next()); }
if (it.hasNext()) { System.out.println(it.next()); }
if (it.hasNext()) { System.out.println(it.next()); }
if (it.hasNext()) { System.out.println(it.next()); }
if (it.hasNext()) { System.out.println(it.next()); }
```

replace repetition with while loop

```
java.util.Collection<String> coll;
coll = new java.util.LinkedList<String>();
coll.add("Fred");
coll.add("Wilma");
coll.add("Pebbles");
coll.add("Dino");
coll.add("Fred");
System.out.println("The collection: " + coll);
java.util.Iterator<String> it;
it = coll.iterator();
while (it.hasNext()) {
        System.out.println(it.next());
}
```

replace repetition with while loop

```
java.util.Collection<String> coll;
coll = new java.util.LinkedList<String>();
coll.add("Fred");
coll.add("Wilma");
coll.add("Pebbles");
coll.add("Dino");
coll.add("Fred");
System.out.println("The collection: " + coll);
java.util.Iterator<String> it;
it = coll.iterator();
while (it.hasNext()) {
   String s = it.next();      // introduce local variable
        System.out.println(s);
}
```

for-each loop – process each element of a collection of elements of type E:

```
for (E item : collection) {
    // do something with item
}
```

Example: to print out each String in coll:

```
for (String s : coll) {
    System.out.println(s);
}
```

while loop

```
java.util.Collection<String> coll;
coll = new java.util.LinkedList<String>();
coll.add("Fred");
coll.add("Wilma");
coll.add("Pebbles");
coll.add("Dino");
coll.add("Fred");
System.out.println("The collection: " + coll);
java.util.Iterator<String> it;
it = coll.iterator();
while (it.hasNext()    ) {
        String s = it.next();
        System.out.println(s);
}
```

for-each loop: no explicit iterator

```
java.util.Collection<String> coll;
coll = new java.util.LinkedList<String>();
coll.add("Fred");
coll.add("Wilma");
coll.add("Pebbles");
coll.add("Dino");
coll.add("Fred");
System.out.println("The collection: " + coll);


for (String s : coll) {

        System.out.println(s);
}
```

for-each loop vs. while loop

```
java.util.Collection<String> coll;
coll = new java.util.LinkedList<String>();
coll.add("Fred");
coll.add("Wilma");
coll.add("Pebbles");
coll.add("Dino");
coll.add("Fred");
System.out.println("The collection: " + coll);
```

```
for (String s : coll) {

        System.out.println(s);
}
```

```
Iterator<String> it;
it = coll.iterator();
while (it.hasNext()) {
  String s = it.next();
  System.out.println(s);
}
```

syntactic sugar

The for-each loop is “syntactic sugar”

- sugar tastes good, but delivers “empty” calories
- in a programming language, something labelled as syntactic sugar is a convenience which does not augment the power of the language

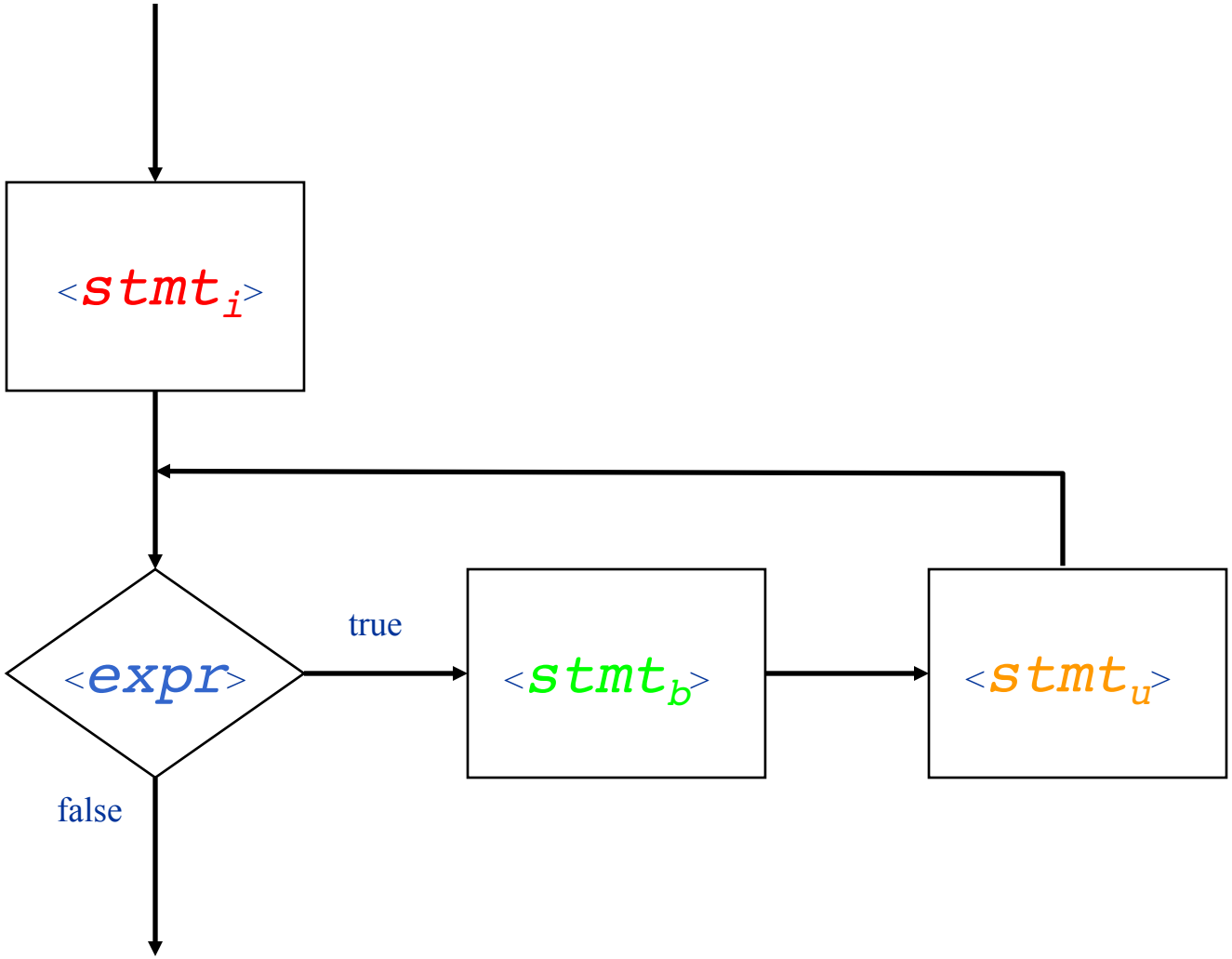
prior to Java 5 the language did not have a for-each loop

any for-each loop can be replaced by an equivalent while loop or for loop

the for-each syntax is a convenience, but doesn’t add to the power of the language

Control structure overview: for statement

for (<stmt$_i$> ; <expr> ; <stmt$_u$>) <stmt$_b$>



EXAMPLE

```
/* Compute the product of the integers 1 through limit (inclusive)
 * Using FOR loop (regular, not enhanced)
 */
public int product(int limit) {
        int answer = 1;
        for (int i=1; i<=limit; i=i+1) {
                answer = answer * i;
        }
        return answer;
}

/* Equivalent WHILE loop
 */
public int productW(int limit) {
        int answer = 1;
        int i=1;
        while (i<=limit) {
                answer = answer * i;
                i=i+1;
        }
        return answer;
}
```

EXAMPLE

```
/* Compute the product of the integers 1 through limit (inclusive)
 * Using FOR loop (regular, not enhanced)
 */
public int product(int limit) {
        int answer = 1;
        for (int i=1; i<=limit; i=i+1) {
                answer = answer * i;
        }
        return answer;
}

/* Equivalent WHILE loop
 */
public int productW(int limit) {
        int answer = 1;
        int i=1;
        while (i<=limit) {
                answer = answer * i;
                i=i+1;
        }
        return answer;
}
```

for-each loop vs. while loop

```
java.util.Collection<String> coll;
coll = new java.util.LinkedList<String>();
coll.add("Fred");
coll.add("Wilma");
coll.add("Pebbles");
coll.add("Dino");
coll.add("Fred");
System.out.println("The collection: " + coll);
```

```
for (String s : coll) {

        System.out.println(s);
}
```

```
Iterator<String> it;
it = coll.iterator();
while (it.hasNext()) {
  String s = it.next();
  System.out.println(s);
}
```

iterator-controlled for loop

```
java.util.Collection<String> coll;
coll = new java.util.LinkedList<String>();
coll.add("Fred");
coll.add("Wilma");
coll.add("Pebbles");
coll.add("Dino");
coll.add("Fred");
System.out.println("The collection: " + coll);


for (Iterator<String> it = coll.iterator(); it.hasNext();) {
    String s = it.next();
    System.out.println(s);
}
```