Project JETT
Java Engagement for Teacher Training

Duke University Department of Computer Science
Association for Computing Machinery
College Board

Who we are

- Faculty
  - Owen Astrachan
  - Robert Duvall
  - Dee Ramm
  - Susan Rodger
- Grad Students
  - Shannon Pollard
  - Jam Jenkins
- Undergrad Students
  - Megan Murphy
  - Andrew Van Kirk
  - Mike Sullivan

Goals of JETT

- ACM’s long-term goal is the creation of a community of computer science teachers, university faculty, undergraduate and graduate students, as well as College Board Lead Teachers.

- The short-term objective is to assist high school computer science teachers in making the AP language switch from C++ to Java for the upcoming academic year.

- ACM is partnering with the College Board and selected universities to provide teachers with on-site workshops as well as remote training via the ACM K-12 Web Repository.

Top ten reasons Java is better than C++

10. `main` still exists, but it really is a void function
9. `No = and ==` confusion, but instead `equals` and `==`
8. Built-in graphics API, don't need CMU, DrawBox, ...
7. Write once/run everywhere, write once/debug everywhere
6. Libraries, libraries, libraries
5. Applets make it easier to show others how clever you are
Top ten reasons continued

4. `java.math.BigInteger` has no overloaded operators

```java
BigInteger count = BigInteger.ZERO;
count = count.add(BigInteger.ONE);
```

3. Type safety means Scheme folk won’t yell as loudly

2. CLASSPATH problems are more easily dealt with than are Turbo 4.5 problems

#1 Reason Java is better than C++

James Gosling

Bjarne Stroustrup

James Gosling

- “Invented” Java
  - First developer, originator,
  - World’s greatest programmer?
  - Canadian


“So, in some sense, vagueness is inescapable. We’re human, and you always have to interpret anybody’s documentation with a certain set of reasonable-person filter to it.”

Bjarne Stroustrup

- “When I was tempted to outlaw a feature [of C++] I personally disliked, I refrained from doing so because I did not think I had the right to force my views on others”

- 1994
  - Grace Murray Hopper Award
- 2002
  - From AT&T to Texas A&M
Some Java Features

- **Platform independence**
  - Source and byte code are the same across platforms
  - Windows, Linux, Mac not all created equal
- **Security**
  - Designed into the language/runtime for code that “travels”
- **Automatic garbage collection**
  - Cannot free storage, cannot control when collection occurs
- **Safety**
  - e.g., error to index past the end of an array, cannot cast objects improperly
- **Simplicity and complexity**
  - Syntactically simple with enormous libraries

Java primitives (not objects)

- **Logical**: boolean
  - No boolean/int conversion or cast
  - while(list) {...}
- **Text**: char
  - Unicode, not part of AP subset
- **Integral**: byte, short, int, long
  - By definition: 8, 16, 32, 64 bits long
- **Floating point**: double, float
  - Use double, float not in the subset

What's a rectangle?

- **Rectangles are everywhere**
  - If you don’t see one, put one around what you do see
- **Familiar from geometry, simple, used as bounding-box**
  - In an object-oriented world, what behavior is there?
  - What’s the state that “makes” a rectangle?
- **How do we construct a rectangle?**
  - Constructors initialize all state so things are reasonable
  - Can have more than one constructor
    - From width, height, from another Rectangle, from nothing

From concept to code via BlueJ

- We know what a rectangle is, how do we write a class so that we can create Rectangles programmatically?
  - What’s the public view, what’s the private view?
- Write something simple, use BlueJ to examine and debug, then grow the class by adding to working code
  - Don’t spend time early on being general if not required
  - We might want polygons, should we use an array of four points for rectangle rather than x,y,width,height?
    - Agile software methodology (formerly light-weight)
- BlueJ is a programming environment that makes it easy to interact with objects rather than programming them
  - Avoid main and avoids input
Adding behavior to SimpleRectangle

- Add simple behavior: perimeter and area
  - What are return types? What are parameters?
  - How do we test these in BlueJ?

- Rectangles are anchored in the plane: add top-left point with integral coordinates.
  - Add a new constructor, provide default values in original

- Add a point to a rectangle so that the rectangle grows minimally to contain the new point: add(int x, int y)
  - Don’t use if, use static methods Math.min and Math.max
  - What are these and where do they come from?

Growing with math

- From original rectangle and a point we grow the rectangle
  - What are possibilities?
  - How can state change?

- What methods are in Math? In subset and in Math?
  - Math.pow, Math.sqrt, Math.abs
  - Math.min, Math.max

- Static methods aren’t invoked via object, but via class
  - What’s the difference?

Static, math, details

- It’s not possible to create a Math object
  - Can’t do it in BlueJ, so have to write methods that use the Math functions
  - The Math constructor Math() is made private in java.lang.Math, what’s the effect of this?

- Static methods are called class methods, differentiated from object methods
  - Don’t act on an object, can’t access state, only static fields

- No other static methods are in the AP subset, but some useful
  - Arrays.sort, Collections.sort, more to come?

How will we test the new method?

- Construct test cases that exercise all scenarios
  - What are these, what are expected results?
  - How can we test that our method matches results?

- We could add accessor functions to SimpleRectangle
  - Return state information to client program
  - Typically we speak of getters and setters or accessors and mutators

- We could write a method to determine if two rectangles are equal and use this to test our code
  - We can’t use == we must use .equals()
When are objects equal?

- For primitive types we test for equality with ==
  - We expect 7 == 7, and x == y if both are 7
- For objects the == check sees if addresses are the same
  - Remember for x == y non-primitive, pointers used!
- All classes inherit equals(...) from Object, must override in subclasses

```java
public boolean equals(Object o) {
    SimpleRectangle s = (SimpleRectangle) o;
    // code here comparing this object to s
}
```

Are we re-inventing the wheel?

- "We must not forget that the wheel is reinvented so often because it is a very good idea; I’ve learned to worry more about the soundness of ideas that were invented only once" — David Parnas, Why Software Jewels are Rare
- There’s a class java.awt.Rectangle that has everything
  - What’s the package that it’s in? How do we access it?
  - If it’s the kitchen sink, are we better with our own?
  - How do we read the API and what is that anyway?
- Why are there public fields? What does Rectangle inherit from? Are rectangle comparable for equality?

Motivating problems?

- Given hundreds of rectangles in some form
  - Find the smallest in area
  - Find the one furthest left, break ties with top-left
  - Find the widest rectangle
  - Return a list of all rectangles that intersect some other one(no duplicates in list)
- What’s a list and how do we make one? What are differences between array and ArrayList?
  - What’s casting all about? Is dynamic sizing important?
- How can we sort Rectangles? See question two above
  - What does it mean to be less than?

What about I/O?

- There is an i in file, there is no i in keyboard
  - Why do we need to get input from the user via the keyboard at a command-line prompt?
    - Show our true age
    - Revel in the past
    - Use Unix tools
- See code in WordCounter for what’s involved in reading from files using two approaches
  - What are drawbacks of EasyReader?
  - What are drawbacks of BufferedReader?
How do we get dimensions from user?

- We can use BlueJ to interact with objects entering parameters
  - Intuitive, no code to write, can still test, doesn’t scale?

- We can use a JOptionPane, see BigFactorial.java
  - Get strings from user, parse strings using static methods in Integer and Double classes (bad data entry problem?)

- We can use a third party reader, e.g., EasyReader
  - Not standard, but built on standard tools

- We can build keyboard entry out of raw java.io code
  - Not very pretty, exceptions caught, ...

Top 10: Choosing array vs ArrayList

10. `a.length` has same number of characters as `a.size()` but doesn’t require using the shift key

9. Too many options: `int[] list` compared to `int list[]`

8. Array initialization with `int[] list = {1,2,3,4,5};`

7. Freedom not to choose:

   ```java
   for(int k=0; k < a.size(); k++) ...
   Iterator it = a.iterator(); while (it.hasNext())...
   ```

Top-10: choosing array vs ArrayList

6. ArrayList, what’s that: a list or an array?

5. “Take away my Integer, but leave me my int” — Puff Daddy

4. You can add a String to an ArrayList, but you can’t get a String out of an ArrayList (well, ok, you can with a cast).

3. `Collections.sort` is stable, `Arrays.sort` isn’t (mostly)

2. `list[k] = list[j] vs. list.set(k, list.get(k))`

1. No method to shuffle an array (Collections.shuffle)

Arrays and the AP subset

- One and two-dimensional arrays in subset
  - Two-dimensional arrays will likely move to AB only
    ```java
    int[][] grid = new int[6][10];
    int rows = int.length;
    int cols = int[0].length;
    ```

- Initialization in subset, e.g., `int[] list = {1,2,3,4,5};`

- No java.util.Arrays methods in subset
  - sort, binarySearch, fill, asList, …
ArrayList and the AP subset

- Inheritance hierarchy (List in java.util) is AB only
  - Iterator and ListIterator are AB only
- Downcast from Object to expected type is in subset
  
  ```java
  list.add(new String("hello"));
  String s = (String) list.get(0);
  ```

- Required methods:
  - size(), get(int), set(int, Object),
  - add(Object), add(int, Object), remove(int)
- NOT required:
  - remove(Object), addAll(Collection), clear()

What is an Iterator?

- What problems do Iterators address?
  - Access elements independently of implementation
  - Client programs written in terms of generic component

  ```java
  public void print(Collection c) {
    Iterator it = c.iterator();
    while (it.hasNext()) {
      System.out.println(it.next());
    }
  }
  ```

- How do you add all elements of Set to a List?

What is an interface?

- Indication to programmers and code that a class implements some specified functions, most likely with requirements on behavior
  - Iterator is an interface: what do we expect from classes that implement the Iterator interface?
  - Comparable: are some objects incomparable? Why?
  - Why isn’t Equatable an interface? Where is .equals()?

- A class can implement multiple interfaces
  - Comparable, Cloneable, Tickable, ...

- Think twice before developing inheritance hierarchy
  - Single inheritance, problems with protected/private data

What is Comparable?

```java
String a = "hello";
String b = "zebra";
int x = a.compareTo(b);  // what values assigned?
int y = b.compareTo(a);
int z = a.compareTo(a);
```

- Contract: compareTo() should be consistent with equals()
  - What’s the simple way to write equals for Comparable?

- See also java.util.Comparator
  - Not in subset, useful for sorting on other criteria
What is Computer Science?

What is the central core of the subject? What is it that distinguishes it from the separate subjects with which it is related? What is the linking thread which gathers these disparate branches into a single discipline? My answer to these questions is simple -- it is the art of programming a computer. It is the art of designing efficient and elegant methods of getting a computer to solve problems, theoretical or practical, small or large, simple or complex. It is the art of translating this design into an effective and accurate computer program.

– Tony Hoare

Tradeoffs

- Programming should be about tradeoffs: programming, structural, algorithmic
  - Programming: simple, elegant, quick to run/to program
    - Tension between simplicity and elegance?
  - Structural: how to structure data for efficiency
    - What issues in efficiency? Time, space, programmer-time
  - Algorithmic: similar to structural issues
- How do we decide which choice to make, what tradeoffs are important?

Guidelines for using inheritance

- Create a base/super/parent class that specifies the behavior that will be implemented in subclasses
  - Some functions in base class may be abstract
    - Subclasses required to implement, or cannot create object
    - Consider using an interface if there’s no default behavior or state to provide
- Inheritance models “is-a” relationship, a subclass is-a parent-class, can be used-as-a, is substitutable-for
  - Standard examples include animals and shapes
    - Square should NOT derive/inherit from rectangle
    - A square is NOT a rectangle in programming terms

Inheritance (language independent)

- First view: exploit common interfaces in programming
  - Streams in C++, Iterator in Java
    - Iterators in STL/C++ share interface by convention/templates
    - Implementation varies while interface stays the same
- Second view: share code, factor code into parent class
  - Code in parent class shared by subclasses
  - Subclasses can override inherited method
    - Can subclasses override and call?
- Polymorphism/late(runtime) binding (compare: static)
  - Actual function called determined when program runs, not when program is compiled
Top 10 reasons for moving AB to Java

10. Things are simpler when everything is a pointer
9. More material to cover with Map and Set, helps use up abundance of free time available in AB courses
8. Just when you thought it was safe to go back in the water, the fish become unbounded
7. ap.exam.QuiltProblem throws StoryTooLong Exception
6. I never understood how templates worked anyway
5. Carpal tunnel syndrome: ArrayList v. apvector; String v string; ArrayStack v. apstack

Reason # 4

```java
void print(const map<string,vector<int> >& m)
{
    map<string,vector<int> >::iterator it = m.begin();
}
```

Garbage Collection

- Allocated memory does not need to be deallocated (delete).
  - For most programmers, who knows best?
  - Cannot prevent, cannot invoke garbage collection
  - Can take steps to free resources, e.g., close files, dispose of Graphics contexts.

- Garbage Collection
  - Concurrent/system thread tracks memory allocation
  - Checks and frees memory no longer needed/accessible
  - Designed to have minimal impact, implementation varies across platforms and JVM implementations

Java Object Basics

- Class is an object factory
  - Class can have (static) methods
  - Defines interface programmers depend on
  - Part of inheritance hierarchy (every class is-an Object)
  - May define getter and setter methods for accessing state
  - Usually defines methods for behavior

- An object is an instance of a class
  - Attributes aka state, usually private, sometimes accessible
  - Behavior invoked by calling methods
  - Allocated by calling new
Top 10 reasons continued

3. In grading AP exams, we can have a usage error for capitalization problems, e.g., the real-estate error

```java
public class Fish
{
    public Location location()
    {
        return myLoc;
    }
}
```

```java
ArrayList list = new ArrayList();
Iterator it = list.iterator();
// code from ArrayList class follows
// public Iterator iterator() {}
```

#2 Reason to switch AB to Java

1. Source code: `java.util.HashMap` and `stl_hashtable.h`

```java
public boolean containsKey(Object key)
{
    Entry tab[] = table;
    if (key != null) {
        int hash = key.hashCode();
        int index = (hash & 0x7FFFFFFF) % tab.length;
        for (Entry e = tab[index]; e != null; e = e.next)
            if (e.hash==hash && key.equals(e.key))
                return true;
    }
    return false;
}
```

```java
size_type count(const key_type& __key) const
{
    const size_type __n = _M_bkt_num_key(__key);
    size_type __result = 0;
    for (const Node* __cur = _M_buckets[__n]; __cur; __cur = __cur->_M_next)
        if (_M_equals(_M_get_key(__cur->_M_val), __key))
            ++__result;
    return __result;
}
```

#1 Reason for using Java in AB

1. No friends, but there is free garbage collection

```
java.util.Collection
```

### Interface for collections of objects

- See API for details, AP subset requires only a few of the methods for subinterfaces List and Set, and for separate interface Map

- Which interface and class should be used? Tradeoffs.
  - Why choose ArrayList vs LinkedList?
  - Why choose HashSet vs TreeSet?

- What about classes, interfaces, and methods not in the subset?
  - Collection interface compared to Collections class

```
Collection.addAll(Collection); // object method
Collections.sort(Collection); // static method
```
Collection hierarchy, tradeoffs?

Collection
- add
- size
- iterator

List
- add
- size
- iterator
- listIterator

ArrayList
- get(int)
- set(int, o)
- add(int, o)
- remove(int)

LinkedList
- getFirst
- getLast
- addFirst
- addLast
- removeFirst
- removeLast

Set
- contains()
- remove()

HashSet
- contains()
- remove()

TreeSet
- contains()
- remove()