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.

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

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
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;
    return width == s.width && height == s.height;
}
```

Java Array basics

- Syntactically similar to C++ arrays
  ```java
  int[] list = new int[10]; // values stored?
  String list2[] = new String[100];
  for(int k=0; k < list.length; k++) {
      list[k] = 3;
  }
  Arrays.fill(list2,"Hello"); // not in subset
  ```
- Once created cannot resize, indexing is range-checked
  - Array object is-an Object via inheritance
  - Default value is 0/null, not in subset

ArrayList basics

- ArrayList stores Objects, not int, not boolean, yes String ...
  - Don't have syntactic sugar of [] indexing
  - Must cast when getting object out of ArrayList
    - Except when treating as an object, e.g., using toString()
  - Can add to end in constant time, in middle with shifting
    - Array grows, use size() method to determine # elements

  ```java
  ArrayList list = new ArrayList();
  list.add(new String("hello"));
  list.add(new String("world"));
  String s = (String) list.get(0); // what is it?
  list.set(0, new String("big"));
  list.add(0, new String("great"));
  ```

Arrays and the AP subset

- One and two-dimensional arrays in subset
  - Two-dimensional arrays will 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
  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
  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?

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
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: 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())...
   ```
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`