CSE 143
Lecture 2

More `ArrayList`; classes and objects

reading: 10.1; 8.1 - 8.7

slides adapted from Marty Stepp and Hélène Martin

http://www.cs.washington.edu/143/

Words exercise, revisited

- Write a program that reads a file and displays the words of that file as a list.
  - Then display the words in reverse order.
  - Then display them with all plural words removed.

Exercise solution (partial)

```
ArrayList<String> allWords = new ArrayList<String>();
Scanner input = new Scanner(new File("words.txt"));
while (input.hasNext()) {
    String word = input.next();
    allWords.add(word);
}

// display in reverse order
for (int i = allWords.size() - 1; i >= 0; i--) {
    System.out.println(allWords.get(i));
}

// remove all plural words
for (int i = 0; i < allWords.size(); i++) {
    String word = allWords.get(i);
    if (word.endsWith("s")) {
        allWords.remove(i);
        i--;
    }
}
```

`ArrayList` of primitives?

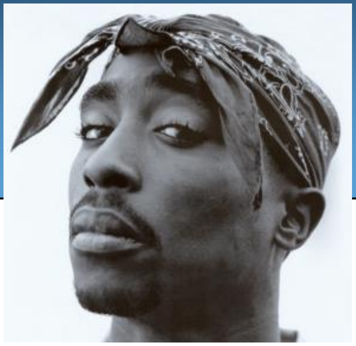
- The type you specify when creating an `ArrayList` must be an object/class type; it cannot be a primitive type.

```
// illegal; int cannot be a type parameter
ArrayList<int> list = new ArrayList<int>();
```

- But we can still use `ArrayList` with primitive types by using special classes called *wrapper* classes in their place.

```
// legal; creates a list of ints
ArrayList<Integer> list = new ArrayList<Integer>();
```

Wrapper classes



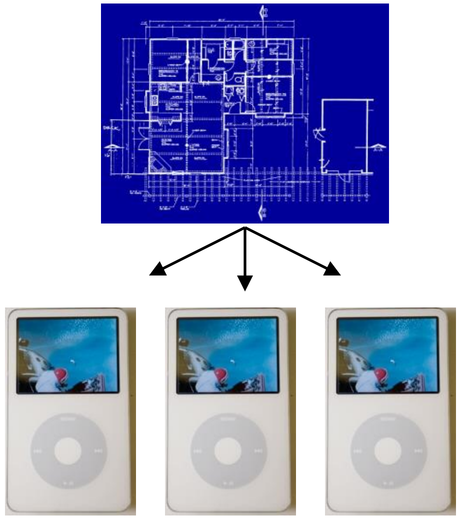
Primitive Type	Wrapper Type
`int`	`Integer`
`double`	`Double`
`char`	`Character`
`boolean`	`Boolean`

- A **wrapper** is an object whose sole purpose is to hold a primitive value.

- Once you construct the list, use it with primitives as normal:

```
ArrayList<Double> grades = new ArrayList<Double>();
grades.add(3.2);
grades.add(2.7);
...
double myGrade = grades.get(0);
```

Classes and objects



- **class**: A program entity that represents:
  - A complete program or module, or
  - A template for a type of objects.
  - (`ArrayList` is a class that defines a type.)

- **object**: An entity that combines **state** and **behavior**.
  - **object-oriented programming (OOP)**: Programs that perform their behavior as interactions between objects.
  - **abstraction**: Separation between concepts and details. Objects provide abstraction in programming.

`BankAccount` exercise

- Suppose we have a class `BankAccount` with the methods:

```
public BankAccount(String name, int id)
public void deposit(double amount)
public void withdraw(double amount)
public double getBalance()
public int getID()
```

- Make each account keep a log of all its transactions.
  - Desired: a `printLog` method that shows all transactions so far.

```
Deposit of $7.82
Withdrawal of $2.55
Deposit of $6.18
```

Objects storing collections

- An object can have an array, list, or other collection as a field.

```
public class Course {
    private double[] grades;
    private ArrayList<String> studentNames;

    public Course() {
        grades = new double[4];
        studentNames = new ArrayList<String>();
        ...
    }
```

- Now each object stores a collection of data inside it.