Java: Base Types

- All information has a type or class designation
- Built-in Types
  - Called: primitive types or base types
    boolean, char, byte, short, int, long, float, double
  - Primarily use: int, double, boolean
- Need to declare before using; defined (created) when declared
  ```java
  int x, y; // declaration only   // some people group declarations at front
           // use
  x = 39;   // use
  y = 3 + x;
  int num = 10; // declaring when first used
  ```
- Each type has its limitations
  - int: whole numbers only, limited range
  - double: implements scientific notation: fractions and wider range
  - boolean: true or false values only
Java: Operators for Base Types

- **Familiar operators available:** +, −, *, /, %
  - For ints, / yields whole number quotient
  - % yields the remainder
  - What are
    - 12/5 ?
    - 13%5 ?
  - What is the meaning of \((2*\text{pancakes} + \text{capacity} -1)/\text{capacity}*5\) ?

- **Also have comparison operators:** <, <=, ==, !=, >=, >
  - Do not use = (assignment) where you mean ==

- **Learn about operator precedence from text (p22)**
  - Most pretty intuitive – well designed language
  - However, when in doubt, use parentheses

- **Most of these operators have no meaning for objects**
Java: Objects

- **Objects are instantiations of a class**
  - Use new operator to create
  - Invokes constructor which initializes object

- **Assume we have class DLicense**
  - (Class names start with capital letter)
  - DLicense john;  // declare
    - (Object names (e.g. john) start with lower case letter)
  - John = new DLicense();  // create
  - DLicense susan = new DLicense( );  //combine
    - Creation may allow or require arguments
    - Depends on constructor(s)
Java: Classes

- **Combine data and function in one package**
  - Use `new` operator to create
  - Invokes constructor which initializes object

- **Data**
  - Store in *instance variables* (fields)
    - May be primitive types or
    - May be objects

- **Methods**
  - Often called functions, subroutines, or procedures
  - Usually do things to and with the instance variables
Java: Methods (Functions)

- Classes usually define one or more methods
  - Structure:
    
    ```java
    access return_type name(parameters) {
        // method body
    }
    ```

- Accessor methods return info
  - Return_type indicates type of info returned
  - Often have empty parameter list

- Mutator methods change state
  - Usually parameters are involved in making the changes
  - Often no info returned, thus return_type is `void`
    - (Object names (e.g. john) start with lower case letter)

- Invoking methods
  - Use: `object.method(params);`
  - Object is implicit parameter
Java: Sample Class

public class DLicense {  
    private String name;
    private String dob;
    private double height;
    public DLicense() { // constructor
        name = "";
        dob = "";
        height = 0.0;
    }
    public DLicense(String nm, String bd){
        name = nm;
        dob = bd;
        height = 0.0;
    }
    // methods to follow
Java: Sample Class (cont)

```java
public String getName() {
    return name;
}
public String getDOB() {
    return dob;
}
public double getHeight() {
    return height;
}
public void setHeight(double ht) {
    height = ht;
}
public String display() {
    return name + " " + height + "\" born: " + dob;
}
```
public static void main(String[] args) {
    DLicense john;
    john = new DLicense();
    john.setName("John Smith");
    john.setHeight(72.0);
    DLicense susan = new DLicense();
    susan.setName("Sue Peggy");
    susan.setHeight(66.5);
    Bud.setHeight(68.0);
    System.out.println(bud.display());
    System.out.println(john.getName());
    System.out.println(susan.getHeight());
}
}
Java Strings

- String is a class
  - Do not need new to create String
    ```java
    String msg = "hello";
    ```
  - String constants (literals) use "", not '
- Can join strings (concatenate) with +
  ```java
  String mail = "John says " + msg;
  ```
- Most common String methods:
  - int length(); // get number of chars in it
  - String substring(int start, int stop); // substring gets part of string
  - int indexOf(String key); // finds loc of key
Java Control of Flow (by example)

- **if**
  
  ```java
  if (a > b) {
      msg = "good";
  }
  ```

- **if-else**
  
  ```java
  if (w.hasMore()) {
      System.out.println(w.next());
      count = count + 1;
  }
  else
      System.out.println("All done");
  ```
Java Control of Flow (by example)

- **while**
  
  ```java
  n = 10;
  while (n > 0){
      System.out.println("down to " + n);
      n = n - 1;
  }
  ```

- **for**
  
  ```java
  for (n = 10; n > 0; n--)
      System.out.println("down to " + n);
  ```

- **do-while**
  
  ```java
  n = 10;
  do {
      System.out.println("down to " + n);
      n = n - 1;
  } while (n > 0);
  ```
What can an Object do (to itself)?

  - Look at java.lang.Object

- `toString()`
  - Used to print (System.out.println) an object, overriding `toString()` can result in 'useful' information being printed, also used in String concatenation: `String s = x + y;`
  - Default is basically a pointer-value

- `equals()`
  - Determines if guts of two objects are the same, must override, e.g., for using `a.indexOf(o)` in ArrayList `a`
  - Default is `==`, pointer equality

- `hashCode()`
  - Hashes object (guts) to value for efficient lookup
Objects and values

- **Primitive variables are boxes**
  - think memory location with value

- **Object variables are labels that are put on boxes**

```java
String s = new String("genome");
String t = new String("genome");
if (s == t) {they label the same box}
if (s.equals(t)) {contents of boxes the same}
```

What's in the boxes? "genome" is in the boxes
Objects, values, classes

- For primitive types: int, char, double, boolean
  - Variables have names and are themselves boxes (metaphorically)
  - Two int variables assigned 17 are equal with ==

- For object types: String, Sequence, others
  - Variables have names and are labels for boxes
  - If no box assigned, created, then label applied to null
  - Can assign label to existing box (via another label)
  - Can create new box using new

- Object types are references or pointers or labels to storage