Welcome

- **Time:** 6:30 to 9:00 then bar til 9:30. No rush, cover what we can.
- **Format:** Talk, (D)emo, (E)xercise, (S)olutions & "more"
- **Agenda:**
  Primitives
  Objects
  Association
  Inheritance
  Interfaces (If time)

Getting Started

- **Code along:**
  1. Connect: Wifi, Meetup conversation, GoogleDoc
  2. Setup: See instructions for Java, Eclipse, Git
  3. Get project: `git clone` , `git pull` to get latest, refresh into IDE
  4. Simple.java (Command line)
  5. Hello.java (Eclipse)
  Topics: package, main, args, public

1. Primitives.

Primitives, Output, Operators, Control flow, Loops

- **Primitive variable actually holds the data.**

  `int i= 7;`

  `int j= i;` // How many ints?

  `double d= 1.23;` // Care re precision!

1. Primitives…

- **Java is strongly typed.** Declare before use. Can only assign to same type.
- `System.out.println( "i= "+ i);    /* Multi-line comment */`
- `System.out.printf( " %d %f %s %n ", i, d, s );` Strings covered later
- **Operators:** The usual arithmetic also `==` `+=` and `++`
- **Choice** `if (a == b) { … } else { … }` Note == not =   likewise && || and / or
- **Loop** `for (int i=0; i<9; i++) { … }`
- **Demo1 then Exercise1**
- **More**
  - Operators: `!= % & | ?`
  - Choice: `switch`
  - Loops: `while`

2. Objects.

Objects, Scanner, String , User defined classes, Array, Setters, Constructors

- **A class** groups together related data (and code) eg Person has `age` and `height`
- **Objects** are "instances of a Class" eg `fred` is a `Person`
- **Object variables** DO NOT hold the data, they are just references to Objects.
- **Create objects** by using `new`

  `Person p = new Person();`

  `Person p2= p;` // How many Persons?

2. Objects…

- **Invoke** the Object’s behavior (code) by using .methodName() eg aScanner.nextInt()
- **String** is special (don’t need `new`) BUT it is still an object, take care re Comparison
- **Array variable** is also an object reference, need `new` to create array

  `int[] ai= new int[7];`

  `for (int x: ai) { … }` // New style for loop
- **User defined objects** and Arrays of (references to!) them

  `class Person { … }`

  `Person[] pa= new Person[7];` // How many Persons?
- **Working with objects:** Setters, Constructors, Comparison, Copies
- **Demo2 then Exercise2**

3. Association between classes

Implement `has a` relationship

- House **has a** Door etc. Gets complex, Draw a UML sketch!
- **Design considerations:**
  Custody: Who creates, owns, moves, removes the Door.
  Ownership: Can I add my Door to your House?
  Sharing: Can 2 houses share same door?
  May be adjoining. Or need methods to move / remove doors
- **Demo3 then Exercise3**

4. Inheritance.

- Accurately **model** our problem domain
  Classes aren't unique, they belong to categories or classifications

- Car **`is a kind of`** Vehicle (dont confuse with `kind of is a`)
  substitutable ie a Car can do everything that a Vehicle can do
  does some things **differently** eg `alertWalkers()`
  does some **extra things** eg `drive()`

- **Ask:** What's **the same** (in Base class), What's **extra**, What's **different** (Override)

- Java supports **Single Inheritance**, can only `extend` one thing, as per real world!
  and **Single Object Inheritance**, all classes automatically extend the root `Object` Class

- **Demo4 then Exercise4**

4. Inheritance...

- **Other syntax details**
  @Override annotation
  `abstract` methods and classes (Give me a Fruit)
  Polymorphism, many forms, base variable can refer to any sub-class object, we dont know or care which "form"
  protected Allow sub-class methods to access
  super() call in constructor (must be first statement), like earlier this() call
  super.aMethod() call in child class method can appear in any seqeuence
- **Demo4b then Exercise4b**

5. Interfaces

- Classify things by **what they can do**, rather than what they "are a kind of"
- Java allows a class to `implements` any number of `interfaces`
- eg some Buildings can be used as a Dwelling (House, Flat), as can some Vehicles (Boat, Motorhome) even though they belong to separate hierarchy / inheritance families.
- **Demo5 then Exercise5**
- **More:** Generality, Decouple for flexibility, DI Spring etc