Introduction to Programming
with Java, for Beginners

Private
Random Class (E.g. OOP class)
Primitive vs. References Type
Stack vs. Heap
Static vs. Dynamic

Encapsulation or Information Hiding
- One of the advantages of OOP is that object need not reveal all of its attributes (data/state) and behavior
- We can hide details of one object from another
- Use modifiers (private/public) to hide information
  - Ideally we make all instance variable(s) private
- Provide methods (query/command) if you want to allow the data to read or written
  - Getter methods to read e.g. getAge()
  - Setter methods to modify e.g. setAge() -> not necessary to provide

Scope Issues with variables

```
public class Dot{
  private int x;
  private int y;
  public Dot(int x, int y){
    this.x = x;  // fixed!!
    this.y = y;
  }
}
```

Local variable `x & y` shadows the instance variable `x & y`

```
public class Dot{
  private int x;
  private int y;
  public Dot(int x, int y){
    this.x = x;  // fixed!!
    this.y = y;
  }
}
```

Solution

```
public class Dot{
  private int x;
  private int y;
  public Dot(int x, int y){
    this.x = x;  // fixed!!
    this.y = y;
  }
}
```

`this` is a reference to the current object
- The object whose constructor/method is being called
- The value of "this" is an object’s heap address

Asking Object about its data directly
- It may be possible to ask a object about its data without querying the object
  - public or no modifier
  - `ObjectName.DataField;`
- But you can prevent such change by making object data `private`
  - E.g. `private int age;`

Example in Dr Java
```
> Student s1 = new Student("Joe",5);
> s1.age
5
> s1.age = 6;
> s1.age
6
```
Random Class

- A class to create Random numbers
- Constructor Summary shows the objects of this type can be created
  - E.g. Random r = new Random();

- Method Summary shows that it can generate random values of types:
  - integers, doubles etc.
  - E.g. r.nextInt(6) – Generate a integer numbers between 0 (inclusive) and 6 (exclusive)

Packages and import Statements

- What is a package?
  - Basically it's a directory that has a collection of related classes
  - E.g. Random Class description contains: java.util.Random
  - Indicating that the Random class code is stored in the directory path java/util/ somewhere on your machine
    - "util", or utility package

- In order to use implemented work, need to tell Java compiler where class is located
  - Use import statement
    - import java.util.Random;
    - Another way is to use the asterisk "wildcard character": import java.util.*
    - Import statement is written outside the class description in a file

Dynamic Variables and Methods

- All instance variables (object data) and methods (object behavior) created without static keyword
  - Note: There is no "dynamic" keyword in Java
  - Dynamic by default

- In general, dynamic refers to things created at "run time" i.e. when the program is running

- Every object gets its own (dynamic) instance variables

- Every object effectively gets its own copy of each dynamic method (i.e. the instructions in the method)
Static Variables with OO class

- **Static** means “pertaining to the class in general”, not to an individual object
- Variable is declared with the `static` keyword outside all methods
- A static variable is *shared* by all instances (if any)
  - All instances may be able read/write it

Use of static variable I

- Global Constants
  - Constants are variable that don’t change
  - Constants are made static because there is no need for more than one copy it
- Example:

```java
class Deck{
    public static final int JACK = 11;
    public static final int QUEEN = 12;
    public static final int KING = 13;
    public static final int SPADE = 1;
    ...
}
```

Use of static variable II

- Providing communication among instances of classes i.e. objects
- In this case using static variable is way of accessing some common resource

Example: Ticket No. Generator

```java
public class Ticket{
    // shared
    private static int numTicketsSold = 0;

    // one per object
    private int ticketNum;

    public Ticket(){
        numTicketsSold = numTicketsSold + 1;
        ticketNum = numTicketsSold;
    }
}
```

**Note:** static variable is used to generate ticketNum and in way keeps track of the number of tickets sold which can be accessed by all objects
Static Methods with OO class

- A method may be declared with the `static` keyword
- Static methods live at class level, not at object level
- Static methods can access static variables and methods, but not dynamic ones
  - How could it choose which one? We have not created any objects yet
- Example:
  ```java
  public static int getNumSold(){
    return numTicketsSold;
  }
  ```

Example: Ticket No. Generator

```java
public class Ticket{
  private static int numTicketsSold = 0; // shared
  private int ticketNum; // one per object

  public Ticket(){
    numTicketsSold = numTicketsSold + 1;
    ticketNum = numTicketsSold;
  }

  public static int getNumberSold() {
    return numTicketsSold;
  }

  public int getTicketNumber() {
    return ticketNum;
  }

  public String getInfo(){
    return "ticket # " + ticketNum + "; " +
    numTicketsSold + " ticket(s) sold.";
  }
}
```

Static Variables & Methods in General

- A static method that is public can be accessed outside class definition
  - `ClassName.methodName(args)`
  - `double result = Math.sqrt(25.0);`
  - `int sold = Ticket.getNumberSold();`

- A static variable that is public may be accessed
  - Using `ClassName.variableName`
    - E.g. `Math.PI, Math.E`
  - Static variables act as global variable i.e. accessible within any static method

When to use static with OOP

- A variable should be static if:
  - It logically describes the class as a whole
  - There should be only one copy of it

- A method should be static if:
  - It does not use or affect the object that receives the message (it uses only its parameters)
Static & Dynamic Rules Recap

- **static** variables and methods belong to the class in general, not to individual objects.
- The absence of the keyword **static** before non-local variables and methods means **dynamic** (one per object-instance).
- A dynamic method can access all dynamic and static variables and methods in the same class.
- A static method can not access a dynamic variable (How could it choose or which one?)
- A static method can not call a dynamic method (because dynamic method might access an instance variable).

Primitive vs. Reference Types

- We’ve seen Java’s 4 **primitive** types: int, double, boolean, char.
- Types other than the primitive types are known as **reference** types.
  - Used for objects.
- Examples of reference variables:
  - ```Student s1; Counter c1; String name;```  
  - Note: String is an object not primitive type.

How the Stack Works

<table>
<thead>
<tr>
<th>DrJava Interactions</th>
<th>Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt; int x;</td>
<td>![Stack Diagram for int x]</td>
</tr>
<tr>
<td>&gt; x = 5;</td>
<td>![Stack Diagram for x = 5]</td>
</tr>
<tr>
<td>&gt; double min = 0.5;</td>
<td>![Stack Diagram for double min]</td>
</tr>
<tr>
<td>&gt; boolean done = false;</td>
<td>![Stack Diagram for boolean done]</td>
</tr>
</tbody>
</table>

Note: Variables are added in the order they are declared.

Reference Type

- The term **reference** is used because it refers to a memory location where the object lives.
  - The variable of reference type is used to access the object.
- The value of variable of reference type is either “null” or a “heap address.”
  - **null** means currently not pointing at any location.
Value of a Reference Variable

Example:
```java
> Counter c1;
> c1
null
> c1 = new Counter();
> c1
Counter@e05ad6
```
- `e05ad6` is location in memory where object that `c1` is pointing resides
  - `e05ad6` hexadecimal (base 16) number
  - This location will differ on your computer
- We don’t have to (and can’t) deal with these hex numbers directly
  - Convenience of using variables

How the Heap Works

<table>
<thead>
<tr>
<th>DrJava Interactions</th>
<th>Stack and Heap</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt; int x = 99;</td>
<td></td>
</tr>
<tr>
<td>&gt; Counter c1;</td>
<td></td>
</tr>
<tr>
<td>&gt; c1</td>
<td></td>
</tr>
<tr>
<td>null</td>
<td></td>
</tr>
<tr>
<td>&gt; c1 = new Counter();</td>
<td>e1</td>
</tr>
<tr>
<td>&gt; c1</td>
<td></td>
</tr>
<tr>
<td>null</td>
<td></td>
</tr>
<tr>
<td>&gt; c1.incrementCount();</td>
<td></td>
</tr>
<tr>
<td>&gt; Counter c2 = new Counter();</td>
<td></td>
</tr>
<tr>
<td>&gt; c2</td>
<td></td>
</tr>
<tr>
<td>Counter@2f996f</td>
<td></td>
</tr>
</tbody>
</table>

String

- A sequence of characters
- A String is a built-in Java object type
- Java provides this type because it's used so frequently
- Examples of String creation:
  ```java
  > String s1 = new String("hello");
  > String s2 = "hello"; // commonly used shortcut
  > s2 + " you!"  
  "hello you!"
  > s2 = "The result is " + 100;
  > System.out.println(s2);  
  "The result is 100"
  ```

Aliases

- Two or more references can point to the same object
  - These references are then known as aliases
- Example (In Dr Java Interactions Pane)
  ```java
  > Student s1 = new Student("Lisa", 5);  
  > s1
  Student@83d8be
  > Student s2 = s1;  
  > s2
  Student@83d8be
  > s1.getAge()  
  5
  > s2.getAge()  
  5
  ```

String

- A sequence of characters
- A String is a built-in Java object type
- Java provides this type because it's used so frequently
- Examples of String creation:
  ```java
  > String s1 = new String("hello");
  > String s2 = "hello"; // commonly used shortcut
  > s2 + " you!"  
  "hello you!"
  > s2 = "The result is " + 100;
  > System.out.println(s2);  
  "The result is 100"
  ```
String (contd..)

<table>
<thead>
<tr>
<th>DrJava Interactions</th>
<th>Stack and Heap</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt; String c1 = new String(&quot;Hill&quot;);</td>
<td>![Diagram of Stack and Heap]</td>
</tr>
</tbody>
</table>

> String c1 = new String("Hill");
> c1
"Hill"

**Note:**
- We do not get heap address of String reference
- Later, when we learn "Inheritance" it will be clear