Lecture Notes
Chapter #9_a
Objects Revisited

1. Java API Classes

- Date, Random, JFrame
- String, StringBuilder, File
- Scanner, PrintWriter
- Etc

2. Immutable Classes & Objects

- immutable object -- an object whose contents cannot be changed once the object is created, e.g., String object
- immutable class – the class of an immutable object, e.g., String class
- requirements
  - all data fields must be private
  - there are NO mutator methods
  - accessor methods cannot return a reference to any mutable data field

```
public class Student
{
        private int id;
    private String name;
        private java.util.Date dateCreated;

public Student(int ssn, String newName)
{
    id = ssn;
    name = newName;
    dateCreated = new java.util.Date( );
}
public int getId( ){ return id; }
public String getName( ){ return name; }
public java.util.Date getDateCreated( )
   { return dateCreated; }
}
```

getDateCreated( ) accessor method returns a reference to the Date object

which allows the time to be modified

Student class in not immutable

```
public class Test
{
   public static void main(String [ ] args)
   {
      Student student = new Student(123456789, "John");
      java.util.Date dateCreated = student.getDateCreated( );
      dateCreated.settime(123456);
   }
}
```

```
public class Circle3_Immutable
{
private double radius;
private static int numberOfObjects = 0;
public Circle3{  numberOfObjects++; }

public Circle3(double newRadius)
{
   radius = newRadius;
   numberOfObjects++;
}

public double getRadius( ){ return radius; }
public static int getNumberOfObjects( ){ return numberOfObjects; }
public double getArea( ) { return radius * radius * Math.PI; }
```

3. Scope of Variables

- local variables – variables declared inside a method
  - scope of local variables: **point of declaration → end of method**
  - local variables may be declared, with the same name, multiple times in different blocks

- class variables – instance or static variables declared inside a class, i.e., data fields
  - scope of class variables: **entire class regardless of where declaration occurred**
    exception: if data field A is initialized based on a reference to data field B, then data field B must be declared before data field A, e.g.,
    private int i;
    private int j = i + 1;
  - class variables may only be declared once in a class
  - class methods may be declared in any order in a class

- if a local variable has the same name as a class variable, the local variable takes precedence and the class variable is hidden

```
class Foo
{
  int x = 0; // instance variable (class variable)
  int y = 0;
  Foo{ }

  void p( )
  {
    int x = 1; // local variable
    System.out.println(x);
    System.out.println(y);
  }
}
```

prints "1" since the local variable takes precedence over a class variable

prints "0" since the class variable is accessible inside the method p( )

- the **this** reference – a reference that refers to the calling object itself
  - a hidden instance variable can be referenced by the keyword **this**
  - a hidden static variable can be referenced by the using the **ClassName.staticVariable** reference

```
public class Foo
{
  int i = 5;
  static double k = 0;
  void setI(int i)
  {
     this.i = i;
  }
  static void setK(double k)
  {
    Foo.k = k;
  }
}
```

```
f1.setI(10);
Foo.setK(7);
```

f1

i = 10

k = 7

this

parameter name "i" is identical to the data field name "i"

```
public class Circle
{
  private double radius;

  public Circle(double radius)
  { this.radius = radius }

  public Circle( ) { this(1.0); }

  public double getArea( )
  {
    return radius * radius * Math,PI;
  }
}
```

parameter name "radius" of the Constructor is identical to the data field name "radius" which necessitates the use of the **this** reference variable

using **this**(1.0) functions as a call to another constructor in the class that can process the enclosed data, i.e., the previous constructor

Constructor with fewer or no arguments can invoke a constructor with more arguments using the this(arg-list) – multiple constructors

4. Class Abstraction & Encapsulation

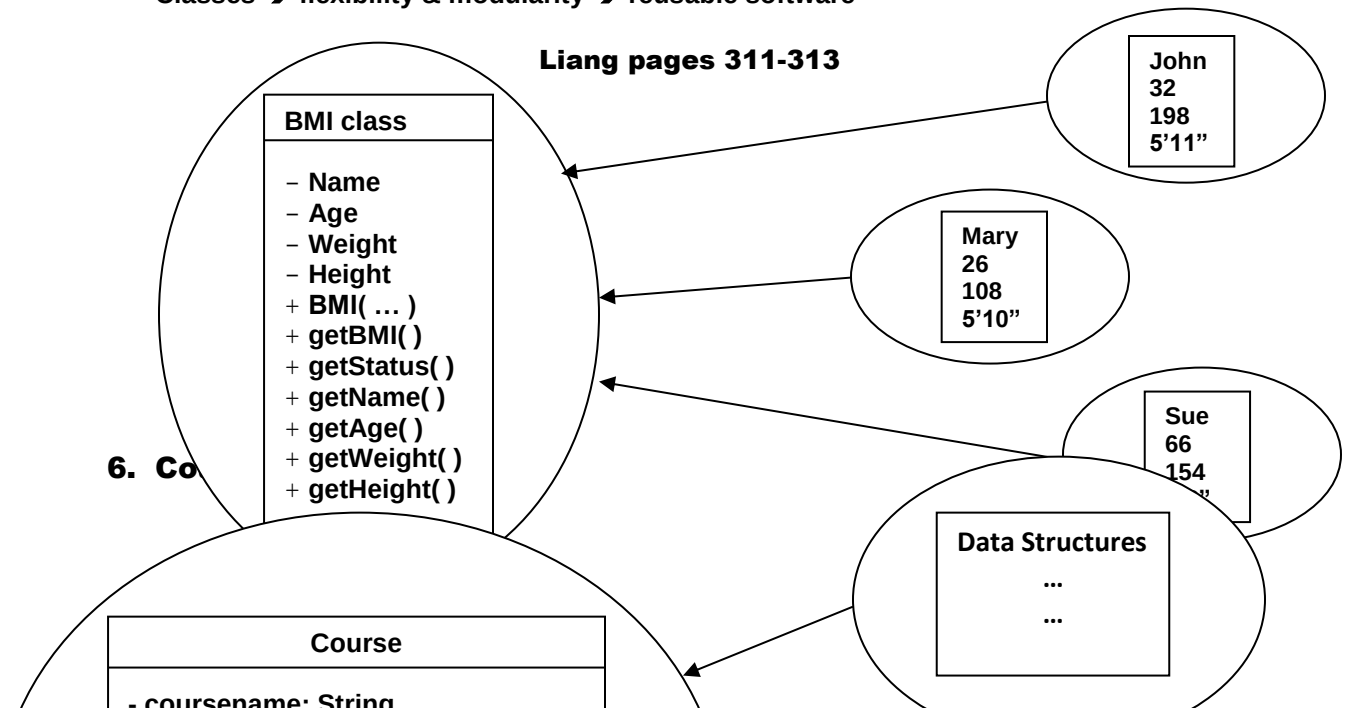
- class abstraction – separation of class implementation from class usage
  - class contract – collection of accessible methods, accessible fields, and a complete description of the expected behavior of the accessible methods and fields
  - class encapsulation – details of the class implementation are encapsulated, i.e., hidden, from the user

- **Liang pages 308-311**
  - **Loan UML Diagram**
  - **Using the Loan class to construct a specific object**
  - **Constructing the Loan class to service multiple users**

5. Object Oriented Design

Classes ➔ flexibility & modularity ➔ reusable software

Liang pages 311-313



6. Co

Liang pages314-315

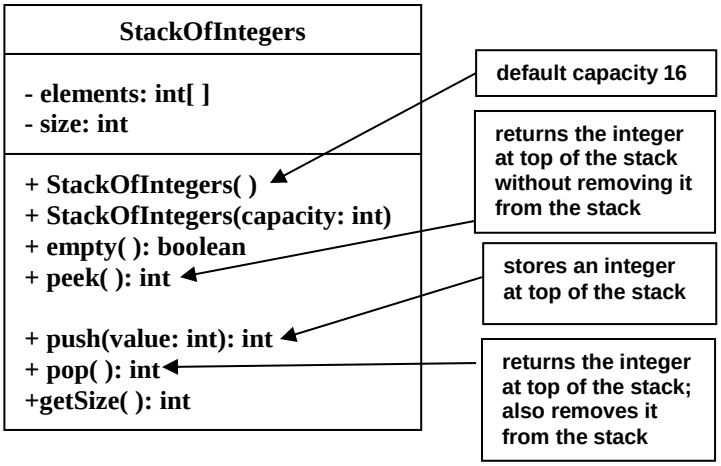
TestCourse.java

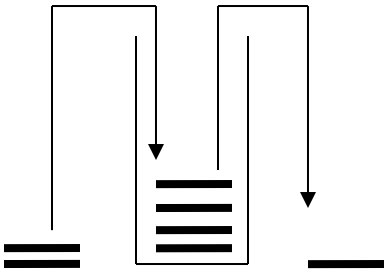
```
public class TestCourse
{
  public static void main(String[ ] args)
  {
    Course course1 = new course("Data Structures);
    Course course2 = new course("Database Systems);

    course1.addStudent("Peter Jones");
    course1.addStudent("Brian Smith");
    course1.addStudent("Anne Kennedy");

    course2.addStudent("Peter Jones");
    course2.addStudent("Steve Smith");

    System.out.println("course1 student count: "
        + course1.getNumberOfStudents( ));
    String[ ] students = course1.getStudents( );
    for(i = 0; i < course1.getNumberOfStudents( ); i++)
        System.out.print(Students[ i ] + ", ");
    System.out.println( );

    System.out.println("course2 student count: "
        + course2.getNumberOfStudents( ));
    String[ ] students = course2.getStudents( );
    for(i = 0; i < course2.getNumberOfStudents( ); i++)
        System.out.print(Students[ i ] + ", ");
    System.out.println( );
  }
}
```

Course.java

```
public class Course
{
   private String courseName;
   private String [ ] students = new String[ 100 ];
   private int numberOfStudents;

   public Course(String courseName);
        { this.courseName = courseName; }

   public void addStudent(String string)
   {
      Students[numberOfStudents] = student;
      numberOfStudents++;
   }

   public String[ ] getStudents( ){ return students; }

   public int getNumberOfStudents( )
                { return numberOfStudents;}

   public String getCourseName( )
                { return courseName;}

   public void dropStudent(String student){      }
}
```

7. Stacks



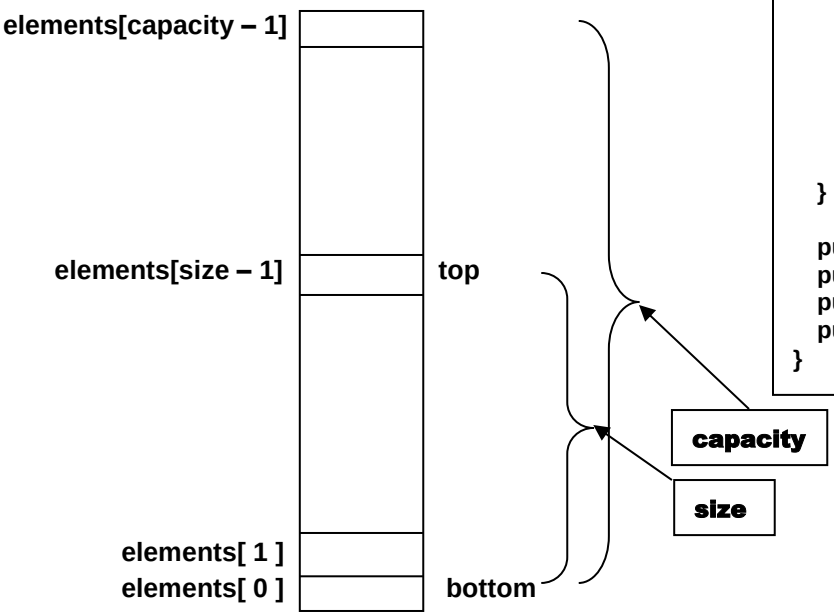


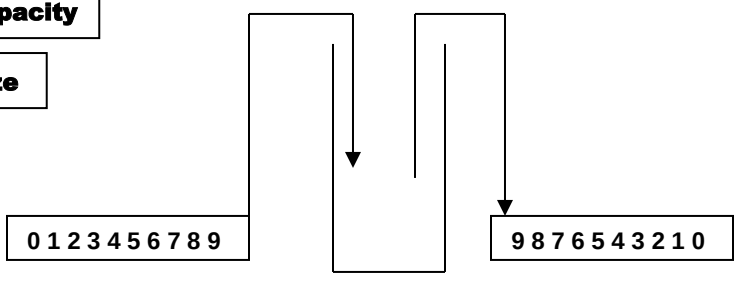
TestStackOfIntegers.java

```
public class TestStackOfIntegers
{
   public static void main(String[ ] args)
   {
      StackOfIntegers stack = new StackOfIntegers( );

      for(int i = 0; i < 10; i++)
            { stack.push( i );}

      while ( !stack.empty( ))
            { System.out.print(stack.pop( ) + " ");
   }
}
```

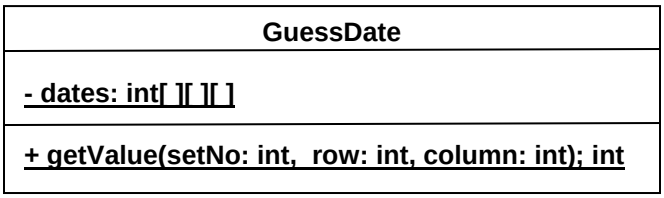
StackOfIntegers.java

```
public class StackOfIntegers
{
   private int[ ] elements;
   private int size;
   public static final int DEFAULT_CAPACITY = 16;

   public StackOfIntegers( )
                  {this(DEFAULT_CAPACITY);}

   public StackOfIntegers(int capacity)
                  { elements = new int[capacity]; }

   public int push(int value)
   {
      if(size >= elements.length)
      {
         int[ ] temp = new int[elements.length*2];
         System.arraycopy(elements, 0, temp, 0,
                              elements.length);
         elements = temp;
      }
      return elements[size++] = value;
   }

   public int pop( ){ return elements[- -size]; }
   public int peek( ){ return elements[size – 1];
   public boolean empty( ){ return size == 0; }
   public int getSize( ){ return size; }
}
```





8. GuessDate Class



UseGuessDateClass.java

```
import java.util.Scanner;

public class UseGuessDateClass
{
   public static void main(String[ ] args)
   {
      int date = 0;
      int answer;
      Scanner input = new Scanner(System.in);
      for(int i = 0; i < 5; i++)
      {
         System.out.println("Birth Date in Set " + (i+1) + "?");
         for(int j = 0; j < 4; j++)
         {
            for(int k = 0; k < 4; k++)
               System.out.print(GuessDate.getValue(i, j, k) + " ");
            System.out.println( );
         }
         System.out.print("\nNo – enter 0; Yes – enter 1: ");
         answer = input.nextInt( );
         if(answer == 1) date += GuessDate.getValue(i, 0, 0);
      }
      System.out.println("Birth Date: " + date);
   }
}
```

Prevent user from creating an object of type GuessDate

GuessDate.java

```
public class GuessDate
{
   public final static int[ ][ ][ ] dates =
   {
     {   { 1, 3, 5, 7 },
         { 9, 11, 13, 15 },
         { 17, 19, 21, 23 },
         { 25, 27, 29, 31 }  },
    {   { 2, 3, 6, 7 },
        { 10, 11, 14, 15 },
        { 18, 19, 22, 23 },
        { 26, 27, 30, 31 }  },
    {   { 4, 5, 6, 7 },
        { 12, 13, 14, 15 },
        { 20, 21, 22, 23 },
        { 28, 29, 30, 31 }  },
    {   { 8, 9, 10, 11 },
        { 12, 13, 14, 15 },
        { 24, 25, 26, 27 },
        { 28, 29, 30, 31 }  },
    {   { 16, 17, 18, 19 },
        { 20, 21, 22, 23 },
        { 24, 25, 26, 27 },
        { 28, 29, 30, 31 }  }
   };

   private GuessDate(  ){ }

   public static int getValue(int setNo, int k; int j)
   {
      return dates[ setNo ][ k ][ j ];
   }
}
```