Java Classes

class MyClass {
    int field1;
    int field2;

    void method1 (int arg1) {
        method body
    }
}

- field1 and field2 are instance variables.
  They don’t exist until a new instance of
  the class is created using new: MyClass m
  = new MyClass(); m.field1 and m.field2 are now created.

- We can also invoke method1:
  m.method1(5).

Slide 5–1

Java Constructors

class Point {
    private int x; private int y;

    public Point (int X, int Y) {
        x = X; y = Y;
    }

    String toString () {
        return "(" + x + ", " + y + ")";
    }
}

- Point is a constructor. It gets called
  whenever a new object gets created:
  Point p = new Point(5,6) creates a
  new point with x=5 and y=6.

- We can print the point to standard
  output like this:
  System.out.println(p). println calls
  Point’s toString-method automatically.
Java Inheritance

class Point {
    protected int x; protected int y;
    public Point (int X, int Y)
        {x = X; y = Y;}  
    String toString ()
        {return "("+x", "+y")";}
}
class ColorPoint extends Point {
    private int z;
    public ColorPoint (int X, int Y, int Z)
        {super(x, y); z = Z; }  
    String toString ()
        {return "("+x", "+y", "+z")";}
}

• ColorPoint is a subclass of Point. It
  adds a new field, z, a new constructor
  (which calls Point’s constructor), and
  overrides toString.

Slide 5–4

The main method

class myClass {
    public static void main (String[] args)
    {
        System.out.println("Hello world!");
    }
}

• The class must be in a file called myClass.java.

Run with java myClass.

class-file myClass.class.

static methods are similar to functions in C.

Slide 5–5

Arrays I

class Point {    ...    class myClass {
    public static void main (String[] args) {
        Point[] a = new Point[5];
        a[0] = new Point(5,6);
        a[15] = new Point(10,4); // Error
    }
}

• Arrays are bounds-checked. Arrays don’t grow. Use
  java.util.Vector for expandable arrays.

Slide 5–6

Arrays II

class myClass {
    public static void main (String[] args) {
        int intArray[] = {1,2,3,4,5};

        char charArray[] = {'a','b','c'};

        String[] stringArray[] = {'A','b','cat'};

    }
}

• Arrays can be initialized as part of the declaration.

Slide 5–7
Multi-Dimensional Arrays

- Write a program which constructs an 8 x 8 integer matrix of random values 0...9.

```java
class Trans {
    public static int[][] random (int len) {
        int[] a = new int[len];
        for (int i = 0; i < len; i++)
            a[i] = (int) (Math.random() * 10.0);
        return a;
    }
    public static void main (String args[])
```.

Slide 5–8

The Class Math

- Standard mathematical functions (trigonometric, exponentiation, etc) are supplied by a standard, built-in class called Math.

```java
public final class java.lang.Math
    extends java.lang.Object {
    public final static double E=...;
    public final static double PI=...;

    public static int max(int a, int b);
    public static int min(int a, int b);
    public static int round(float a);

    public static double abs(double a);
    public static double ceil(double a);
    public static double floor(double a);
    public static double exp(double a);
    public static double log(double a);
    public static double pow(double a, double b);
    public static double sqrt(double a);
    public static double max(double a, double b);
    public static double min(double a, double b);
    public static double random();
    public static double acos(double a);
    public static double asin(double a);
    public static double atan(double a);
    public static double atan2(double a, double b);
    public static double cos(double a);
    public static double sin(double a);
    public static double tan(double a);
```

Interfaces

```java
public interface Man {
    public eat ();
    public sleep ();
}
public interface Cowperson {
    public drink (Whiskey w);
    public fight (Cowboy who);
    public ride (Horse h);
}
public class Cowboy
    implements Man, Cowperson {
    public eat () { ... };
    public sleep () { ... };
    public drink (Whiskey w) { ... };
    public fight (Cowboy who) { ... };
    public ride (Horse h) { ... };
```

Slide 5–9

Slide 5–10

Slide 5–11
The Rational Class I

- A rational number is represented by two integers, the denominator and the numerator. We need operations for creating a new rational number, performing arithmetic on two rational numbers, and converting a rational number to a string (for printing).

```java
class Rational {
    Rational (int Denom, int Numer) {
        D = Denom; N = Numer;
    }
    public Rational add (Rational B) {
        return new Rational (D*B, N + M*B, D, M, B);}
    public Rational sub (Rational B) {
        return new Rational (D*B, N - M*B, D, M, B);}
    public Rational mul (Rational B) {
        return new Rational (D*B, N*M, D, M, B);}
    public Rational div (Rational B) {
        return Rational (D*B, N*M, D, M, B);}
    public String toString () {
        return Integer.toString (D) + " + Integer.toString (N);
    }
}
```

The Rational Class II

```java
class Rat {
    private static void test (Rational a, String Op, Rational b, Rational R) {
        System.out.println (a + Op + b" = " + R);
    }
    public static void main (...) {
        Rational a = new Rational (3, 5);
        Rational b = new Rational (1, 5);
        test(a, " + ", b, a.add(b));
        test(a, " - " , b, a.sub(b));
        test(a, " * " , b, a.mul(b));
        test(a, " / " , b, a.div(b));
    }
}
```

Slide 5–12

The Rational Class III

```java
private int D, M, N;
Rational (int Denom, int Num) {
    D = Denom; M = Num;
}
public Rational add (Rational B) {
    return new Rational (D*B, N + M*B, D, M, B);}
public Rational sub (Rational B) {
    return new Rational (D*B, N - M*B, D, M, B);}
public Rational mul (Rational B) {
    return new Rational (D*B, N*M, D, M, B);}
public Rational div (Rational B) {
    return Rational (D*B, N*M, D, M, B);}
public String toString () {
    return Integer.toString (D) + " + Integer.toString (N);
}
```

Slide 5–13

### Primitive Types and Operators

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>8-bit signed integer</td>
</tr>
<tr>
<td>short</td>
<td>16-bit signed integer</td>
</tr>
<tr>
<td>int</td>
<td>32-bit signed integer</td>
</tr>
<tr>
<td>long</td>
<td>64-bit signed integer</td>
</tr>
<tr>
<td>float</td>
<td>32-bit floating-point number</td>
</tr>
<tr>
<td>double</td>
<td>64-bit floating-point number</td>
</tr>
<tr>
<td>char</td>
<td>16-bit Unicode characters</td>
</tr>
<tr>
<td>boolean</td>
<td>Can hold true or false</td>
</tr>
</tbody>
</table>

- There are no unsigned integer types.
- Unlike C, `boolean` is distinct from `int`, with its own set of operators:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Operation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>Negation</td>
<td>!a</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>Conditional AND</td>
<td>a &amp;&amp; b</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>?:</td>
<td>Conditional</td>
<td>a ? e1 : e2</td>
</tr>
</tbody>
</table>

Slide 5–14

Slide 5–15
### Bits and Pieces

- Java is Garbage Collected. You allocate a new object using `new`; when that object is no longer reachable it will be automatically reclaimed by the Java runtime system.

- Fields and local variables can be initialized in the declaration:
  ```java
  class Employee {
      protected int age = 21;
  }
  ```
  If there is no explicit initialization, the runtime system will initialize pointers to `null`, integers to `0`, etc.

- Java has no `goto`-statement.
### Libraries

- Java has a bunch of standard libraries.
  - `java.util.*` has classes that implement various types of collections, and the like.

```java
import java.util.*;
class my {
    public static void main (String args[]) {
        Vector myVector = new Vector();
        for(int i=0; i<100; i++)
            myVector.addElement(new Integer(i));
        Enumeration e = myVector.elements();
        while (e.hasMoreElements()) {
            Integer I = (Integer)e.nextElement();
            int i = I.intValue();
            System.out.println(i);
        }
    }
}
```

---

### Exceptions I

- Code that can throw an exception can be enclosed in `try-catch` blocks. Note that the brackets are *not* optional:

```java
try {
    trouble();
}
catch(anException e) {
    System.out.println(e.getMessage());
}
catch(anotherException e) {
    System.out.println(e.getMessage());
}
```

---

### Exceptions II

- Methods that throw exceptions (or who call methods that throw exceptions) must list these exceptions in the header:

```java
public void trouble() throws Exception {
    throw new Exception("big trouble!");
    ...
}
```

- A `finally`-clause contains code that will be executed whether the exception is thrown or not:

```java
try {  
    ...
} catch(anException e) {  
    ...
} finally {  
    this will always be executed
}
```