Data Types

• Primitive data types

<table>
<thead>
<tr>
<th>Type</th>
<th>Bits</th>
<th>Minimum Value</th>
<th>Maximum Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>8</td>
<td>-128</td>
<td>127</td>
</tr>
<tr>
<td>short</td>
<td>16</td>
<td>-32768</td>
<td>32767</td>
</tr>
<tr>
<td>int</td>
<td>32</td>
<td>-2,147,483,648</td>
<td>2,147,483,647</td>
</tr>
<tr>
<td>long</td>
<td>64</td>
<td>-9,223,372,036,854,775,808</td>
<td>9,223,372,036,854,775,807</td>
</tr>
<tr>
<td>float</td>
<td>32</td>
<td>1.40129846432481707e-45</td>
<td>3.40282346638528860e+38</td>
</tr>
<tr>
<td>double</td>
<td>64</td>
<td>4.94065645841246544e-324</td>
<td>1.79769313486231570e+308</td>
</tr>
<tr>
<td>boolean</td>
<td>1</td>
<td>true / false</td>
<td></td>
</tr>
<tr>
<td>char</td>
<td>16</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Wrapper Class

• Each primitives has a corresponding class.
• String is an object rather than a primitive.

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Wrapper Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>Byte</td>
</tr>
<tr>
<td>short</td>
<td>Short</td>
</tr>
<tr>
<td>int</td>
<td>Integer</td>
</tr>
<tr>
<td>long</td>
<td>Long</td>
</tr>
<tr>
<td>float</td>
<td>Float</td>
</tr>
<tr>
<td>double</td>
<td>Double</td>
</tr>
<tr>
<td>boolean</td>
<td>Boolean</td>
</tr>
<tr>
<td>char</td>
<td>Character</td>
</tr>
<tr>
<td>String</td>
<td>String</td>
</tr>
</tbody>
</table>

Wrapper Class

• “Wrap” the primitive data type into an object
• Providing various functionalities
• Part of the java.lang package, which is imported by default
• Use of wrapper class
  - Creating an object
    ```java
    int x = 25;                     // x is a variable.
    Integer y = new Integer(33);   // y is an object.
    ```
  - Accessing value
    ```java
    int z = x + y;                  // wrong!
    int z = x + y.intValue();       // OK!
    ```
  - Other operations
    ```java
    int x = Integer.parseInt("1234");
    String s = Integer.toString(1234);
    ```
**String**

- A sequence of characters
  - An object, but often regarded as primitive data type
  - Java provides the **String** class to create and manipulate strings.

- Creating string
  ```java
  String greeting = "Hello world!";
  ```
  ```java
  char[] helloArray = { 'h', 'e', 'l', 'l', 'o', '.' };
  ```
  ```java
  String helloString = new String(helloArray);
  ```

- String length
  ```java
  String palindrome = "Dot saw I was Tod";
  int len = palindrome.length();
  ```

- String concatenation
  ```java
  String palindrome = "Dot saw I was Tod";
  ```
  ```java
  String helloString = new String(helloArray);
  ```

**Arrays**

- A data structure holding a group of variables under a single identifiers
  - `byte[] anArrayOfBytes;`
  - `int[] anArrayOfInts;`
  - `long[] anArrayOfLongs;`
  - `float[] anArrayOfFloats;`
  - `double[] anArrayOfDoubles;`
  - `boolean[] anArrayOfBooleans;`
  - `char[] anArrayOfChars;`
  ```java
  String anArrayOfStrings = "Hello, world!";
  ```

- `a = new int[2];` // array declaration without assignment
  ```java
  int b = {1, 2, 3, 4}; // array declaration with assignment
  ```
  ```java
  int bLength = b.length; // get the length of an array
  ```

**Operators**

**Arithmetic Operators**

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Example (A=10, B=20)</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Adds values on either side of the operator</td>
<td>A + B will give 30</td>
</tr>
<tr>
<td>-</td>
<td>Subtracts right hand operand from left hand operand</td>
<td>A - B will give -10</td>
</tr>
<tr>
<td>*</td>
<td>Multiplies values on either side of the operator</td>
<td>A * B will give 200</td>
</tr>
<tr>
<td>/</td>
<td>Divides left hand operand by right hand operand</td>
<td>B / A will give 2</td>
</tr>
<tr>
<td>%</td>
<td>Divides left hand operand by right hand operand and returns remainder</td>
<td>B % A will give 0</td>
</tr>
<tr>
<td>++</td>
<td>Increase the value of operand by 1</td>
<td>B++ gives 21</td>
</tr>
<tr>
<td>--</td>
<td>Decrease the value of operand by 1</td>
<td>B-- gives 19</td>
</tr>
</tbody>
</table>

**Relational operators**

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Example (A=10, B=20)</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>Checks if the value of two operands are equal or not</td>
<td>(A == B) is not true.</td>
</tr>
<tr>
<td>!=</td>
<td>Checks if the value of two operands are equal or not</td>
<td>(A != B) is true.</td>
</tr>
<tr>
<td>&gt;</td>
<td>Checks if the value of left operand is greater than the value of right operand</td>
<td>(A &gt; B) is not true.</td>
</tr>
<tr>
<td>&lt;</td>
<td>Checks if the value of left operand is less than the value of right operand</td>
<td>(A &lt; B) is true.</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Checks if the value of left operand is greater than or equal to the value of right operand</td>
<td>(A &gt;= B) is not true.</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Checks if the value of left operand is less than or equal to the value of right operand</td>
<td>(A &lt;= B) is true.</td>
</tr>
</tbody>
</table>
Operators

• Bitwise operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Example (A=60, B=13)</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;</td>
<td>Binary AND Operator</td>
<td>(A &amp; B) will give 12 which is 0000 1100</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Binary OR Operator</td>
</tr>
<tr>
<td>^</td>
<td>Binary XOR Operator</td>
<td>(A ^ B) will give 49 which is 0011 0001</td>
</tr>
<tr>
<td>~</td>
<td>Binary Ones Complement Operator</td>
<td>(~A ) will give -61 which is 1100 0011 in 2's complement form due to a signed binary number.</td>
</tr>
<tr>
<td>&lt;&lt;</td>
<td>Binary Left Shift Operator</td>
<td>A &lt;&lt; 2 will give 240 which is 1111 0000</td>
</tr>
<tr>
<td>&gt;&gt;</td>
<td>Binary Right Shift Operator</td>
<td>A &gt;&gt; 2 will give 15 which is 1111</td>
</tr>
<tr>
<td>&gt;&gt;&gt;</td>
<td>Shift right zero fill operator</td>
<td>A &gt;&gt;&gt;2 will give 15 which is 0000 1111</td>
</tr>
</tbody>
</table>

• Logical operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Example (A=true, B=false)</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;&amp;</td>
<td>Called Logical AND operator</td>
<td>(A &amp;&amp; B) is false</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>!</td>
<td>Called Logical NOT Operator</td>
<td>!(A &amp;&amp; B) is true</td>
</tr>
</tbody>
</table>

• Assignment operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>Simple assignment operator</td>
<td>C = A + B will assign value of A + B into C</td>
</tr>
<tr>
<td>+=</td>
<td>Add AND assignment operator</td>
<td>C += A is equivalent to C = C + A</td>
</tr>
<tr>
<td>-=</td>
<td>Subtract AND assignment operator</td>
<td>C -= A is equivalent to C = C - A</td>
</tr>
<tr>
<td>*=</td>
<td>Multiply AND assignment operator</td>
<td>C *= A is equivalent to C = C * A</td>
</tr>
<tr>
<td>/=</td>
<td>Divide AND assignment operator</td>
<td>C /= A is equivalent to C = C / A</td>
</tr>
</tbody>
</table>

char assignGrade(int testscore)
{
    int testscore = 76;
    char grade;
    if(testscore >= 90)
        grade = 'A';
    ... else if(testscore >= 60)
        grade = 'D';
    else
        grade = 'F';
    return grade;
}

Control Flow

- **switch** statement

```java
void DisplayMonthString (int month)
{
    String month_str;
    switch (month) {
        case 1: month_str = "January";
                break;
        case 2: month_str = "February";
                break;
        ...
        default: month_str = "InvalidMonthNumber";
                 break;
    }
    System.out.println(month_str);
}
```

Loop

- **for** statement

```java
// return {"January", "February", ..., "December"}
String[] month_strs = GetMonthStrings();
for (int i = 0; i < month_strs.length; i++)
{
    System.out.println(month_strs[i]);
}
```

- **while** statement

```java
// return {"January", "February", ..., "December"}
String[] month_strs = GetMonthStrings();
int i = 0;
while(i < month_strs.length)
{
    System.out.println(month_strs[i]);
    i++;
}
```

- **do-while** statement

```java
// return {"January", "February", ..., "December"}
String[] month_strs = GetMonthStrings();
int i = 0;
do
{
    System.out.println(month_strs[i]);
    i++;
} while(i < month_strs.length)
```
Branching

- break, continue and return statements

```java
String padString(String str)
{
    while(1)
    {
        if (str.length() >= 10)
            break; // get out of the loop
        else {
            str += " ";
            continue; // perform one more iteration
        }
    }
    return str; // return a padded string
}
```

Java Keywords

<table>
<thead>
<tr>
<th>abstract</th>
<th>boolean</th>
<th>break</th>
<th>byte</th>
<th>case</th>
<th>catch</th>
<th>char</th>
</tr>
</thead>
<tbody>
<tr>
<td>class</td>
<td>const</td>
<td>continue</td>
<td>default</td>
<td>do</td>
<td>double</td>
<td>else</td>
</tr>
<tr>
<td>extends</td>
<td>final</td>
<td>finally</td>
<td>float</td>
<td>for</td>
<td>goto</td>
<td>if</td>
</tr>
<tr>
<td>implements</td>
<td>import</td>
<td>instanceof</td>
<td>int</td>
<td>interface</td>
<td>long</td>
<td>native</td>
</tr>
<tr>
<td>new</td>
<td>package</td>
<td>private</td>
<td>protected</td>
<td>public</td>
<td>return</td>
<td>short</td>
</tr>
<tr>
<td>static</td>
<td>super</td>
<td>switch</td>
<td>synchronized</td>
<td>this</td>
<td>throw</td>
<td>throws</td>
</tr>
<tr>
<td>transient</td>
<td>try</td>
<td>void</td>
<td>volatile</td>
<td>while</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Java Output

- Formats

```java
System.out.print(<string or var1> + <string or var2>..);
System.out.println(<string or var1> + <string or var2>..);
```

```java
public class OutputExample1
{
    public static void main (String[] args)
    {
        int num = 123;
        System.out.println("Good-night grace!");
        System.out.print(num);
        System.out.println("num=\" + num);
    }
}
```

Java Constant

- Constant

- Constants are like variables in that they have a name and store a certain type of information but unlike variables they CANNOT change.

```java
final int SIZE = 1000;
```
public class Date
{
    // data fields
    private int day;
    private String month;
    private int year;

    // constructors
    public Date()
    {
        day = 1; month = "January"; year = 2013;
    }
    public Date(int d, String m, int y)
    {
        day = d; month = m; year = y;
    }
}

Only class members can access.
Anyone can use this class.
Anyone can call this constructor.
Anyone can call this method.
Calling accessor method

public class TestDate
{
    public static void main(String[] args)
    {
        Date date1 = new Date();  
        Date date2 = new Date(date1);
        Date date3 = new Date(11, "September", 2013);

        System.out.println("date2: "+date2.getMonth()+" "+date2.getDay()+" "+date2.getYear());

        date3.setMonth("October");  
        System.out.println("date3: "+date3.getMonth()+" "+date3.getDay()+" "+date3.getYear());
    }
}

Class hierarchy
• Class hierarchy
  ▪ The "Object" class, defined in the java.lang package, defines and implements behavior common to all classes.
  ▪ Many classes derive directly from "Object", other classes derive from some of those classes, and so on, forming a hierarchy of classes.
Superclass (Base Class)

```java
public class Bicycle {
    // the Bicycle class has three fields
    public int cadence;
    public int gear;
    public int speed;

    // the Bicycle class has two constructors
    public Bicycle() {
        speed = 0;
    }

    public Bicycle(int startCadence, int startSpeed, int startGear) {
        gear = startGear;
        cadence = startCadence;
        speed = startSpeed;
    }
}
```

Subclass (Derived Class)

```java
public class MountainBike extends Bicycle {
    // the MountainBike subclass adds one field
    public int seatHeight;

    // the MountainBike subclass has one constructor
    public MountainBike(int startHeight, int startCadence, int startSpeed, int startGear) {
        super(startCadence, startSpeed, startGear);
        seatHeight = startHeight;
    }

    // the MountainBike subclass adds one method
    public void setHeight(int newValue) {
        seatHeight = newValue;
    }
}
```

Inheritance

- Properties
  - A subclass inherits all the `public` and `protected` members (fields, methods, and nested classes) from its superclass.
  - Constructors are not members, so they are not inherited.
  - New fields, which are not in the superclass, can be declared in the subclass.
  - You can write a new instance method in the subclass that has the same signature as the one in the superclass, thus `overriding` it.
  - You can write a new static method in the subclass that has the same signature as the one in the superclass, thus `hiding` it.
  - You can declare new methods in the subclass that are not in the superclass.
  - You can write a subclass constructor that invokes the constructor of the superclass, either implicitly or by using the keyword `super`.
Many Pre-Created Classes

• Rule of thumb
  ▪ Before writing new program code to implement the features of your program you should check to see if a class has already been written with the features that you need.

• The Java API is Sun Microsystems's collection of pre-built Java classes:
  ▪ [http://java.sun.com/javase/7/docs/api/](http://java.sun.com/javase/7/docs/api/)

• Java tutorial
  ▪ [http://docs.oracle.com/javase/tutorial/](http://docs.oracle.com/javase/tutorial/)

An Example: Integer Wrapper Class

<table>
<thead>
<tr>
<th>Table 1. Methods of the Integer Wrapper Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Method</td>
</tr>
<tr>
<td>Constructors</td>
</tr>
<tr>
<td>Integer(i)</td>
</tr>
<tr>
<td>Integer(s)</td>
</tr>
<tr>
<td>Class Methods</td>
</tr>
<tr>
<td>parseInt(s)</td>
</tr>
<tr>
<td>toString(i)</td>
</tr>
<tr>
<td>Instance Methods</td>
</tr>
<tr>
<td>byteValue()</td>
</tr>
<tr>
<td>doubleValue()</td>
</tr>
<tr>
<td>floatValue()</td>
</tr>
<tr>
<td>intValue()</td>
</tr>
<tr>
<td>longValue()</td>
</tr>
<tr>
<td>shortValue()</td>
</tr>
<tr>
<td>toString()</td>
</tr>
</tbody>
</table>