1 Primitive types

Java provides eight built-in (primitive) data types:

- **int**: the integer type with range $-2,147,483,648$ to $2,147,483,647$ (size: 4 bytes).
  
  **Example**: 5, -1, 0, 0777 (octal), 0xA12D (hexadecimal). Integer literals starting with a leading zero are octal numbers, while integer literals having the prefix 0x are hexadecimal numbers.

- **byte**: A single byte with range -128 to 127.
  
  **Example**:
  
  ```java
  byte b1 = 1;
  byte b2 = 5;
  byte b3 = (byte) (b1 + b2); // cast is required!
  ```

- **short**: an integer type with range $-32768$ to $32767$ (size: 2 bytes)

- **long**: an integer type with range $-2^{63}$ to $(2^{63} - 1)$ (size: 8 bytes).
  
  **Example**: 12L, 06712L (octal), 0xA12D (hexadecimal).

- **float**: A floating point type with range about $-3.4E38$ to $3.4E38$ and 6-7 significant figures of accuracy (size: 4 bytes).
  
  **Example**:
  
  ```java
  float x = 1.23f;
  ```

- **double**: A double precision floating point type with size about $-1.7E308$ to $1.7E308$ and 15 significant figures of accuracy (size 8 bytes).
  
  **Example**:
double y = 44.2;

- **char**: the type used to represent text characters in Unicode encoding (size: 2 bytes).
  
  Example:
  ```
  char grade = 'A';
  ```

- **boolean**: the data type used to represent true/false conditions (size: 1 byte).
  
  Example:
  ```
  boolean b = true;
  ```

2 Limited accuracy of floating point numbers

Because of the way that floating point numbers are represented, a number in a program will only be approximated:

- Number 123,456.789012 cannot be represented by a `float`, as `float`s can hold up to six or seven significant figures.

- Floating point numbers may contain inaccuracies which can mount up (rounding errors).
  
  Example:
  ```
  double f = 4.35;
  System.out.println(100*f); // prints 434.99999999999994
  ```

3 Casts

A type cannot be explicitly assigned to another type, if the two types are incompatible. To do the actual assignment a `cast` is required.

Example:

```
float r = 1.28f;
int m = (int) r; // if cast is omitted, compiler error will be reported!
```

- Some numeric conversions do not need a cast. A smaller range type can be assigned to a larger range directly. E.g. a `byte` to `int`, an `int` to `long`, or `long` to `float`.

- An assignment that potentially might loses data, e.g. assigning a `float` to `int`, must use a cast.
4 Reading Input - The Scanner class

The Scanner class can be used to parse primitive types and strings from input (including from the keyboard).

It contains a number of nextXXX methods including:

- nextDouble(): reads a double.
- nextLine(): reads a line (until the user presses “enter”).
- nextInt(): reads an int.

4.1 Example:

```java
import java.util.*;

public class KeyboardInputExample {
    public static void main(String args[]) {
        Scanner sc = new Scanner(System.in); // create a Scanner object

        System.out.print("Please enter your age: ");
        int age = sc.nextInt(); // read int user input
        System.out.println("You are " + age + " years old!");

        System.out.print("Now enter a double: ");
        double doubleNumber = sc.nextDouble();
        System.out.println("You entered: ") + doubleNumber + \\
    }
}
```

5 Conditionals - The if statement

```
if (expression)
    statement1
else
    statement2
```

or

```
if (expression)
    statement
else if (expression)
    statement
else if (expression)
    statement
else
    statement
```
Statements can be compound statements (i.e. blocks). A statement is an expression, e.g. 
x = 5, or i++, followed by a semicolon. Therefore, the following are statements:

x = 5;
i++; 

Braces { and } are used to group together statements (and also declarations). These are known as compound statements or blocks.

Note that if statements can be nested, i.e. they can be put within another if statement.

Example

Consider the BankAccount example described in the previous set of notes. The withdraw() method should be modified, so that a penalty fee is charged to the account, if it becomes overdrawn.

```java
/**
   A bank account has a balance that can be changed by
   deposits and withdrawals.
*/
public class BankAccount
{
    final int OVERDRAFT_PENALTY = 50;
    private double balance;

    /**
     Constructs a bank account with a zero balance.
     */
    public BankAccount() {
        balance = 0;
    }

    /**
     Constructs a bank account with a given balance.
     @param initialBalance the initial balance
     */
    public BankAccount(double initialBalance) {
        balance = initialBalance;
    }

    /**
     Deposits money into the bank account.
     @param amount the amount to deposit
     */
    public void deposit(double amount) {
        double newBalance = balance + amount;
    }
}```
balance = newBalance;
}

/**
   * Withdraws money from the bank account.
   * @param amount the amount to withdraw
   */
public void withdraw(double amount) {
    if (amount <= balance) {
        double newBalance = balance - amount;
        balance = newBalance;
    } else
        balance = balance - amount - OVERDRAFT_PENALTY;
}

/**
   * Gets the current balance of the bank account.
   * @return the current balance
   */
public double getBalance() {
    return balance;
}

public static void main(String[] args) {
    BankAccount b1 = new BankAccount(10);
    b1.withdraw(20.0);
    double newBalance = b1.getBalance();
    System.out.println("New balance is: "+ newBalance);
}

When the program is run, it displays:

New balance is: -60.0

6 Comparing Objects

The double equals sign == should be avoided to compare objects, as it compares the object addresses rather than the object contents.

- To compare the contents of two objects, the equals() method has to be defined and used.
- The library class String has implemented the equals() method, so as to compare the contents of its object argument with the contents of the String object that it is invoked on.
Example:

```java
public class ObjectComparison {
    public static void main(String[] args) {
        String s1 = new String("abcd");
        String s2 = new String("abcd");

        if (s1 == s2) // address comparison
            System.out.println("s1 == s2 evaluates to: true");
        else
            System.out.println("s1 == s2 evaluates to: false");

        if (s1.equals(s2)) // contents comparison
            System.out.println("s1.equals(s2) is: true");
        else
            System.out.println("s1.equals(s2) is: false");
    }
}
```

When the above program is run, it displays:

```
s1 == s2 evaluates to: false
s1.equals(s2) is: true
```

Note, that if in the above program the two lines in which `s1` and `s2` are assigned are replaced with:

```java
String s1 = "abcd";
String s2 = "abcd";
```

then the comparison of `s1 == s2` would evaluate to true. This is because JVM maintains a pool of `String` literals in order to optimise performance, and avoid the duplication of a `String` object which is represented by a `String` literal.

Thus, when executing the line `String s2 = "abcd"`, the JVM will look into the string pool and see that there exists a literal "abcd". Therefore it will reuse the existing object in the pool, instead of creating a second object "abcd" and add it in the pool. This is illustrated in Figure 1.

7 The null keyword

The null value is predefined in Java to indicate that an object variable (reference) has not been set to a value.
Figure 1: The Java Virtual Machine maintains a pool of string literals. If the **new** keyword is not used during the creation of a **String** object, the JVM will reuse a **String** object with the same characters from the pool, assuming that such an object is already in the pool.

**Example:**

```java
import java.util.Scanner;

class Book {
    public String colour;
}

class NullTest {
    public static void main(String[] args) {
        /* need to create a NullTest object to access its instance field colour */
        Book b = new Book();

        if (b.colour == null) {
            System.out.println("colour in b is unset");
        } else {
            System.out.println("colour of object b is: " + b.colour);
        }
        Scanner sc = new Scanner(System.in);
        b.colour = sc.next(); // read a string from the keyboard

        System.out.println("colour of object b is: " + b.colour);
    }
}
```

Note that as a matter of good object oriented programming practice, the instance fields of a class (such as `colour` above), should be made **private**. In such a case, `getField()`,
setField() public methods should be defined, to access the field for reading and writing respectively.

In the example above, if colour was declared as private, the following methods should be provided:

```java
String getColour() {
    return colour;
}

void setColour(String newColour) {
    colour = newColour;
}
```

8 The switch statement

Used as an alternative to multiple else-if statements, to make a multi-way decision.

Syntax:

```
switch (expression) {
    case const-expr: statements
    case const-expr: statements
    case const-expr: statements
    default: statements
}
```

The value following the keyword case has to be a constant. A variable cannot be used as the value.

If a case matches the expression, the execution starts at that case. The statements of that case and all of the statements in the following cases will be executed. This implies, that if it is desired to execute only the statements of the matching case, a break statement should follow the statements of that case.

The optional default is executed, when none of the other cases are satisfied.

Example:

```java
import java.util.*;

public class SwitchExample {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a day number: ");
        int day = sc.nextInt();
        switch (day) {
            case 1: System.out.println("Sunday");
            case 2: System.out.println("Monday");
            case 3: System.out.println("Tuesday");
            case 4: System.out.println("Wednesday");
            case 5: System.out.println("Thursday");
            case 6: System.out.println("Friday");
            case 7: System.out.println("Saturday");
        }
    }
}
```
break;
case 2: System.out.println("Monday");
break;
case 3: System.out.println("Tuesday");
break;
case 4: System.out.println("Wednesday");
break;
case 5: System.out.println("Thursday");
break;
case 6: System.out.println("Friday");
break;
case 7: System.out.println("Saturday");
break;
default: System.out.println("The number does not correspond to a day!");
}
}

9 Coding style recommendations

The following guidelines should be followed when writing Java programs:

- Class names should start with an uppercase letter.
  
  In the case that the class name is composite (i.e. it consists of more than one word), every single word should start with an uppercase letter, e.g. BankAccount, CombatFlightSimulator, ComplexNumber.

- Method names should start with a lowercase character. However, if the method’s name consists of more than one word, all words subsequent to the first one, must start with an uppercase character.
  
  Examples of methods are: withdraw(), depositMoney(), getBalance(), setCoordinatesOrigin().

- Variables names should be chosen so that they indicate what the variable stands for.
  
  If a variable name consists of more than one word, then the successive words should either be separated by an underscore, or their first character (except for the first word) be made uppercase.
  
  Examples of variables names are: date, number, complex_number, complexNumber, day_of_year, dayOfYear.

- Code should be documented with comments which clarify the way that the implementation works. However, obvious comments such as the one below should be avoided:

  // Do not make comments such as the one below!
  int i; // declare an integer variable

- Blocks of code should be indented properly. Indentation makes clear where a block starts and where it finishes. For example:
if (a == 1) {
    x = 5; // notice the indentation in the statements of this block
    y++;  
}
else if (a == 2) {
    x = 7; // notice the indentation in the statements of this block
    y--;  
}

The statements x = 5; and y++ are executed only if the condition a == 1 is true. If this is not the case, the condition a == 2 is evaluated. The statements x = 7 and y-- will be executed only if that second condition evaluates to true.

if and else have the same indentation, because they are alternatives. Either the block within if or the block within else if will be executed (or none of the two), but both of the blocks will never be executed. This is because the condition in else will be evaluated, only if the condition of the first if is not satisfied.