Lecture 6
• Last time
  – Math.sqrt(x)
Importing methods in the Java library

- A **package** contains **classes** with methods that solve a **related problem**.
- The **standard Java library** is named **java**.

![Diagram of the Java library and its packages and classes.](image)
• Each class inside the Java library has a unique class path name in the form:

  java.PackageName.ClassName
Using classes in the Java library

- Use the class path.

```java
public class ClassPath1 {
    public static void main(String[] args) {
        double x = 1.234;
        double r;

        r = java.lang.Math.sin(x);
        System.out.print("Sin of " + x);
        System.out.println(" = " + r);
    }
}
```
• Import class.

```java
import java.lang.Math;

// After the import clauses, you can use the class
// by its name, without the entire classPath

// This program can now use all things defined inside
// the class java.lang.Math without the classPathName

public class MyProgram
{
    public static void main(String[] args)
    {
        double a;

        a = Math.sqrt(2.0); // Save computed value in variable

        System.out.println(a); // You can print the saved value later
    }
}
```
Importing all classes in a package

```java
import java.lang.Math;
import java.lang.Double;
import java.lang.Integer;
...
import java.util.Scanner;
import java.util.Stack;
...
```

```java
import java.lang.*; // import all classes in java.lang package
import java.util.*; // import all classes in java.util package
```
java.lang

- *All classes* in the *java.lang* package are *automatically included* in every *Java program* (the *Java compiler* is programmed to do this)
Invoking (using) methods in the Java library

```java
public class Math {
    public static void main(String[] args) {
        double a, b, c;

        a = 2.0;
        b = Math.sqrt(a);
        c = Math.pow(a, 0.5);

        System.out.print("a = ");
        System.out.println(a);
        System.out.print("Sqrt of a = ");
        System.out.println(b);
        System.out.print("a^0.5 = ");
        System.out.println(c);
    }
}
```
Reading input from the console input

```java
import java.util.Scanner;

public class MyProgram {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        double x;
        ...
        x = in.nextDouble();
        ...
    }
}
```

1. **import Scanner class**
2. **Construct Scanner object**
3. **Define variable to receive value**
4. **read input**
```java
import java.util.Scanner; // Import Scanner class (contains methods for reading keyboard input)

public class Abc2
{
    public static void main(String[] args)
    {
        double a, b, c, x1, x2; // Define 5 variable

        Scanner in = new Scanner(System.in); // Construct a Scanner object

        a = in.nextDouble(); // Read in next number and store in a
        b = in.nextDouble(); // Read in next number and store in b
        c = in.nextDouble(); // Read in next number and store in c

        x1 = (-b - Math.sqrt(b*b - 4*a*c)) / (2*a);
        x2 = (-b + Math.sqrt(b*b - 4*a*c)) / (2*a);

        System.out.print("a = ");
        System.out.println(a);
        System.out.print("b = ");
        System.out.println(b);
        System.out.print("c = ");
        System.out.println(c);
        System.out.print("x1 = ");
        System.out.println(x1);
        System.out.print("x2 = ");
        System.out.println(x2);
    }
}
```
import java.util.Scanner; // Import Scanner class (contains methods for reading keyboard input)

public class A bc 2 
{
    public static void main(String[] args)
    {
        double a, b, c, x1, x2; // Define 5 variable
        Scanner in = new Scanner(System.in); // Construct a Scanner object

        System.out.print("Enter a = "); // ****** Prompt message
        a = in.nextDouble(); // Read in next number and store in a
        System.out.print("Enter b = ");
        b = in.nextDouble(); // Read in next number and store in b
        System.out.print("Enter c = ");
        c = in.nextDouble(); // Read in next number and store in c

        x1 = (-b - Math.sqrt(b*b - 4*a*c)) / (2*a);
        x2 = (-b + Math.sqrt(b*b - 4*a*c)) / (2*a);

        System.out.print("a = ");
        System.out.println(a);
        System.out.print("b = ");
        System.out.println(b);
        System.out.print("c = ");
        System.out.println(c);
        System.out.print("x1 = ");
        System.out.println(x1);
        System.out.print("x2 = ");
        System.out.println(x2);
    }
}
Converting (Casting)

- To obtain a higher accuracy (= more significant digit of accuracy), we need to combine more memory cells.

- When a Java program needs to use higher accurate numbers, it will not only use more memory, but it will also take a longer time to complete.
Floating point numbers

• Single precision floating point numbers.

  - uses 4 consecutive bytes of memory as a single 32 bit memory cell
  - A single precision floating point variable can represent a floating point number:
    - in range of from $-10^{38}$ to $10^{38}$
    - and with about 7 decimal digits accuracy

• Double precision floating point numbers.

  - uses 8 consecutive bytes of memory as a single 64 bit memory cell
  - A double precision floating point variable can represent a floating point number:
    - in range of from $-10^{308}$ to $10^{308}$
    - and with about 15 decimal digits accuracy
Casting operation = a type conversion operation

(float) --- convert to the single precision floating point representation
(double) --- convert to the double precision floating point representation

```java
public class Casting01 {
    public static void main(String[] args) {
        float x; // Define single precision floating point
        double y; // Define double precision floating point

        x = 3.1415927f; // f denotes "float"

        y = (double) x; // **** convert to double representation

        System.out.print("Original single precision x = ");
        System.out.println(x);
        System.out.print("Converted double precision y = ");
        System.out.println(y);

        x = (float) y; // **** convert to float representation

        System.out.print("Re-converted single precision x = ");
        System.out.println(x);
    }
}
```
• Priority of casting operator.
Lost of accuracy while casting

```java
public class Casting02 {
    public static void main(String[] args) {
        float x;           // Define single precision floating point
        double y;          // Define double precision floating point

        y = 3.14159265358979;  // A "double" typed value
        x = (float) y;        // **** convert to float representation

        System.out.print("Original double precision y = ");
        System.out.println(y);
        System.out.print("Converted single precision x = ");
        System.out.println(x);

        y = (double) x;       // **** convert to double representation

        System.out.print("Re-converted double precision y = ");
        System.out.println(y);
    }
}
```
### Conversion Accuracy

<table>
<thead>
<tr>
<th>Low Accuracy</th>
<th>High Accuracy</th>
<th>Low Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>float</td>
<td>double</td>
<td>float</td>
</tr>
</tbody>
</table>

- **Conversion from float to double:**
  - 3.1415927 → 3.1415927410125732 → 3.1415927
  - **No loss in accuracy**

- **Conversion from double to float:**
  - 3.14159265358979 → 3.1415927 → 3.1415927410125732
  - **HIGH loss in accuracy**
public class Overflow1{
    public static void main(String[] args)
    {
        double d;  // range: $-10^{308}$ .. $10^{308}$
        float f;   // range: $-10^{38}$ .. $10^{38}$

        d = 3.1415e100; // In range of "double", out of range of "float"
        f = (float) d;  // Overflow !!!

        System.out.print("d = ");
        System.out.println(d);
        System.out.print("f = ");
        System.out.println(f);
    }
}

public class Overflow{
    public static void main(String[] args)
    {
        int x, y, z;
        long a, b, c;

        x = 1000000;
        y = 3000;

        z = x * y;       // 3,000,000,000 is outside the range of int
        System.out.println(z);

        a = 1000000;
        b = 3000;

        c = a * b;       // 3,000,000,000 is within the range of long
        System.out.println(c);
    }
}
Safe and unsafe conversion operations

- **Safe conversion** = a conversion from one representation (encoding) to another representation (encoding) where there is no (or very little) loss in accuracy.
- **Unsafe conversion** = a conversion from one representation (encoding) to another representation (encoding) where there is significant loss in accuracy.

- double ⇒ float is a unsafe conversion
- float ⇒ double is a safe conversion
Expressions containing values of different types

• A computer can only operate on data of the same data type

• In order to perform any operation on two values of differing types, the computer must:
  • convert one of the types into the other type
  • Perform the operation on the value (now of the same type)
Automatic conversion between \textit{float} types

- If either operand in a (binary) arithmetic operation is of type \textit{double}, the other operand is converted to \textit{double}.

In other words:

\begin{verbatim}
float + double (automatic) ⇒ double + double
double + float (automatic) ⇒ double + double
\end{verbatim}

- If float \textbf{value} is assigned to a double \textbf{variable}, the float \textbf{value} is converted to \textit{double}.

In other words:

\begin{verbatim}
double variable = float value (automatic) ⇒ double variable = double value
\end{verbatim}

```c
float a = 2.5;
double b = 3.4, c;

c = a + b; // a (float typed) is first converted
// to a double type
// Then the addition is performed.
```
The *general rule* for automatic type conversion

- **Assignment statement:**

  ```java
  variable = expression;
  type1 = type2
  ```

- The *assignment operator* `==` in Java performs *safe* conversions from `type2`⇒`type1` *automatically*.

- If `type1` is a *higher accuracy type* than `type2`, then:
  - the `type2` value is *automatically converted* to `type1` before the assignment statement is executed.
  - (Because the conversion was *safe*)

- If `type1` is a *lower accuracy type* than `type2`, then:
  - the assignment statement is *not allowed*
  - You *must* use an *casting operator* to make the assignment statement valid.
Safe conversions:

\([0 .. 65535]\)

\([a .. b] = \text{the range of value of a given type}\)

\([-128 .. 127]\)

\([-32768 .. 32767]\)

\([-2147483648 .. 2147483647]\)

\([-9223372036854775808 .. 9223372036854775807]\)

\([-1.8 \times 10^{308} .. 1.8 \times 10^{308}]\)

\([-3.4 \times 10^{38} .. 3.4 \times 10^{38}]\)
Unsafe conversions:

double → float → long → int → short → byte

char