Chapter 3  Numerical Data

CS 180
Sunil Prabhakar
Department of Computer Science
Purdue University
Objectives

This week we will study

- Numeric data types
- Arithmetic operators and expressions
- Standard classes
  - Math
  - GregorianCalendar
  - DecimalFormat
  - Input and output classes
Although the primary type of data in OOP is objects, there is another important data type. These are called **primitive types**. In contrast, objects are **reference data types**. There is no class for primitive data. Data in an object must be either another object, or primitive data.

There are three main primitive types:

- Numeric
- Character
- Logic
As with objects, primitive data values are accessed using identifiers.

The declaration specifies the **type** of the identifier
- The type of an object is given by its class
- Primitive data have one of the following types:
  - Numeric: byte, short, int, long, float, double
  - Character: char
  - Logic: boolean

Unlike objects, primitive data items are created upon declaration
- No need to call `new` in order to allocate the space.
Variables
Variables

Data items are also called *variables*.
Variables

- Data items are also called *variables*.
- A variable has three properties:
  - A memory location to store the value,
  - The type of data stored in the memory location, and
  - The name used to refer to the memory location.
Variables

- Data items are also called *variables*.
- A variable has three properties:
  - A memory location to store the value,
  - The type of data stored in the memory location, and
  - The name used to refer to the memory location.

- Sample declaration:
  ```
  int x;
  ```
Data items are also called *variables*. A variable has three properties:

- A memory location to store the value,
- The type of data stored in the memory location, and
- The name used to refer to the memory location.

Sample declaration:

```c
int x;
```

When the declaration is made, memory space is allocated to store *integer* values; `x` is a reference for this space.
Variables

- Data items are also called *variables*.
- A variable has three properties:
  - A memory location to store the value,
  - The type of data stored in the memory location, and
  - The name used to refer to the memory location.
- Sample declaration:
  ```
  int x;
  ```
  When the declaration is made, memory space is allocated to store *integer* values; `x` is a reference for this space.
- It is a “variable” since its value can be changed during program execution. Similarly, object identifiers are also variables.
Assignment Statements

- We assign a value to a variable using an *assignment statement*.
- The syntax is
  
  `<variable> = <expression> ;`

- We will discuss expressions later. For now, they can be numeric values or numeric variables.
- Examples:
  
  ```
  int i, j;
  i = 5;
  j = i;
  ```

- This assignment is different than the assignment to object identifiers.
Primitive Data: Declaration & Assignment

```c
int i, j;
i = 5;
j = 8;
```
Primitive Data: 
Declaration & Assignment

```c
int i, j;
i = 5;
j = 8;
```
Primitive Data: Declaration & Assignment

```plaintext
int i, j;
i = 5;
j = 8;
```

```
int i, j;
i = 5;
j = 8;
```

Memory is allocated.
Primitive Data: Declaration & Assignment

```c
int i, j;
i = 5;
j = 8;
```

Memory is allocated.

Values are stored in those locations.
Primitive Data: Declaration & Assignment

```
int i, j;
i = 5;
j = 8;
```

Memory is allocated.

Values are stored in those locations.
Primitive Data Assignment

```c
int i;
i = 5;
i = 85;
```
Primitive Data Assignment

```plaintext
int i;
i = 5;
i = 85;
```
Primitive Data Assignment

```c
int i;
i = 5;
i = 85;
```

Memory is allocated.
Primitive Data Assignment

```c
int i;
i = 5;
i = 85;
```

Memory is allocated.
The value 5 is stored in i.
Primitive Data Assignment

```c
int i;
i = 5;
i = 85;
```

Memory is allocated.
The value 5 is stored in i.
Primitive Data Assignment

```c
int i;
i = 5;
i = 85;
```

Memory is allocated.
The value 5 is stored in i.
The value 85 is stored in i. Old value is lost.
Object Assignment

```java
Customer customer;
customer = new Customer();
customer = new Customer();
```
Object Assignment

```java
Customer customer;
customer = new Customer( );
customer = new Customer( );
```

The identifier `customer` is allocated.
Object Assignment

```java
Customer customer;
customer = new Customer();
customer = new Customer();
customer
```

The identifier `customer` is allocated.
Object Assignment

```java
Customer customer;
customer = new Customer();
customer = new Customer();
```

The identifier `customer` is allocated.

The reference to the first object is stored in `customer`. 

The reference to the first object is stored in `customer`. 

: Customer

The identifier `customer` is allocated.
Object Assignment

```java
Customer customer;
customer = new Customer();
customer = new Customer();
customer = new Customer();
```

The identifier `customer` is allocated.

The reference to the first object is stored in `customer`.

The reference to the first object is stored in `customer`.

: `Customer`
The identifier `customer` is allocated.

The reference to the first object is stored in `customer`.

```java
Customer customer;
customer = new Customer();
customer = new Customer();
```
The identifier `customer` is allocated.

The reference to the first object is stored in `customer`. The old reference is lost.

The reference to the second object is stored in `customer`. The old reference is lost.
Assigning objects

```java
Customer cust1, cust2;
cust1 = new Customer();
cust2 = cust1;
```
Assigning objects

```java
Customer cust1, cust2;
cust1 = new Customer();
cust2 = cust1;
```

The identifiers are allocated.
Assigning objects

```java
Customer cust1, cust2;
cust1 = new Customer();
cust2 = cust1;
```

The identifiers are allocated.

The reference to the object is stored in `cust1`.

The diagram shows the allocation of identifiers and the reference stored in `cust1`.
Assigning objects

Customer cust1, cust2;
cust1 = new Customer();
cust2 = cust1;

The identifiers are allocated.

The reference to the object is stored in cust1.
Assigning objects

```java
Customer cust1, cust2;
cust1 = new Customer();
cust2 = cust1;
```

- The identifiers are allocated.
- The reference to the object is stored in `cust1`.
- The reference stored in `cust1` is copied to `cust2`.

The diagram shows the process of object assignment in Java.
Assigning Primitive Data

```c
int i, j;
i = 5;
j = i;
```
Assigning Primitive Data

```c
int i, j;
i = 5;
j = i;
```
Assigning Primitive Data

```c
int i, j;
i = 5;
j = i;
```

Memory is allocated.

---

**PURDUE UNIVERSITY**
Assigning Primitive Data

```c
int i, j;
i = 5;
j = i;
```

Memory is allocated.

The value stored in `i` is copied to `j`. 

```
i  5
j
```
Assigning Primitive Data

```c
int i, j;
i = 5;
j = i;
```

Memory is allocated.

![Diagram showing initial values of i and j]

The value stored in i is copied to j.
Assigning Primitive Data

```c
int i, j;
i = 5;
j = i;
```

Memory is allocated.

The value stored in `i` is copied to `j`.
Really the same

Customer cust1, cust2;
cust1 = \textbf{new} Customer();
cust2 = cust1;

: Customer
Really the same

```java
Customer cust1, cust2;
cust1 = new Customer();
cust2 = cust1;
```
Really the same

```java
Customer cust1, cust2;
cust1 = new Customer();
cust2 = cust1;
```
Really the same

Customer cust1, cust2;
cust1 = new Customer();
cust2 = cust1;

cust1

: Customer

cust2

The value stored in cust1 is copied to cust2.
The value happens to be a reference to an object.
Hence reference type vs. primitive type.
Numerical Data Types

- There are six numerical data types: byte, short, int, long, float, and double.
- Sample variable declarations:
  ```
  int i, j, k;
  float numberOne, numberTwo;
  long bigInteger;
  double bigNumber;
  ```
- At the time a variable is declared, it also can be initialized. For example,
  ```
  int count = 10, height = 34;
  ```
- Otherwise, it MAY be given a default value.

Try it out!
# Numeric Data Types

The various data types differ in the precision of the values they can hold.

<table>
<thead>
<tr>
<th>Type</th>
<th>Content</th>
<th>Default Value</th>
<th>Size (bytes)</th>
<th>Minimum Value</th>
<th>Maximum Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>Integer</td>
<td>0</td>
<td>1</td>
<td>-128</td>
<td>127</td>
</tr>
<tr>
<td>short</td>
<td>Integer</td>
<td>0</td>
<td>2</td>
<td>-32768</td>
<td>32767</td>
</tr>
<tr>
<td>int</td>
<td>Integer</td>
<td>0</td>
<td>4</td>
<td>-2147483648</td>
<td>2147483647</td>
</tr>
<tr>
<td>long</td>
<td>Integer</td>
<td>0</td>
<td>8</td>
<td>-9.22337E+18</td>
<td>9.22337E+18</td>
</tr>
<tr>
<td>float</td>
<td>Real</td>
<td>0.0</td>
<td>4</td>
<td>-3.40282347 x 10^{38}</td>
<td>3.40282347 x 10^{38}</td>
</tr>
<tr>
<td>double</td>
<td>Real</td>
<td>0.0</td>
<td>8</td>
<td>-1.7977 x 10^{308}</td>
<td>1.7977 x 10^{308}</td>
</tr>
</tbody>
</table>
## Arithmetic Operators

The following table summarizes the arithmetic operators available in Java.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Java Operator</th>
<th>Example</th>
<th>Value (x=10, y=7, z=2.5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition</td>
<td>+</td>
<td>x + y</td>
<td>17</td>
</tr>
<tr>
<td>Subtraction</td>
<td>-</td>
<td>x - y</td>
<td>3</td>
</tr>
<tr>
<td>Multiplication</td>
<td>*</td>
<td>x * y</td>
<td>70</td>
</tr>
<tr>
<td>Division</td>
<td>/</td>
<td>x / y</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>x / z</td>
<td>4.0</td>
</tr>
<tr>
<td>Modulo division</td>
<td>%</td>
<td>x % y</td>
<td>3</td>
</tr>
</tbody>
</table>

---

Purdue University
### Arithmetic Operators

The following table summarizes the arithmetic operators available in Java.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Java Operator</th>
<th>Example</th>
<th>Value (x=10, y=7, z=2.5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition</td>
<td>+</td>
<td>x + y</td>
<td>17</td>
</tr>
<tr>
<td>Subtraction</td>
<td>-</td>
<td>x - y</td>
<td>3</td>
</tr>
<tr>
<td>Multiplication</td>
<td>*</td>
<td>x * y</td>
<td>70</td>
</tr>
<tr>
<td>Division</td>
<td>/</td>
<td>x / y</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>x / z</td>
<td>4.0</td>
</tr>
<tr>
<td>Modulo division</td>
<td>%</td>
<td>x % y</td>
<td>3</td>
</tr>
</tbody>
</table>

This is integer division where the fractional part is truncated.
Arithmetic expressions
Arithmetic expressions

- An arithmetic expression is composed of numeric values, numeric variables, and operators.
An arithmetic expression is composed of numeric values, numeric variables, and operators.

For example, given: `int i, j;`

```
  i + 3
  (i + 2*(j-i))
  -i + j
```
Arithmetic expressions

- An arithmetic expression is composed of numeric values, numeric variables, and operators.
- For example, given:
  ```c
  int i, j;
  i + 3
  (i + 2*(j-i))
  -i + j
  ```
- Expressions can be used to assign values:
Arithmetic expressions

- An arithmetic expression is composed of numeric values, numeric variables, and operators.
- For example, given: `int i, j;`
  
  \[
  i + 3 \\
  (i + 2*(j-i)) \\
  -i + j
  \]
- Expressions can be used to assign values:
  
  \[
  i = j + 3;
  \]
Arithmetic expressions

- An arithmetic expression is composed of numeric values, numeric variables, and operators.
- For example, given: `int i, j;`
  - `i + 3`
  - `(i + 2*(j-i))`
  - `-i + j`
- Expressions can be used to assign values:
  - `i = j + 3;`
  - Take the value of `j`, add 3 to it and assign that value to `i`. 
Arithmetic Expression
Arithmetic Expression

How is the following expression evaluated?
Arithmetic Expression

How is the following expression evaluated?

\[ x + 3 \times y \]
Arithmetic Expression

How is the following expression evaluated?

\[ x + 3 \times y \]

Answer: \( x \) is added to \( 3\times y \).

Arithmetic Expression

How is the following expression evaluated?

\[ x + 3 \times y \]

Answer: \( x \) is added to \( 3\times y \).

We determine the order of evaluation by following *precedence rules*. 
Arithmetic Expression

- How is the following expression evaluated?
  \[ x + 3 \times y \]

  Answer: \( x \) is added to \( 3 \times y \).

- We determine the order of evaluation by following *precedence rules*.

- Evaluation is in order of precedence.
Arithmetic Expression

- How is the following expression evaluated?
  \[ x + 3 \times y \]

Answer: \( x \) is added to \( 3 \times y \).

- We determine the order of evaluation by following precedence rules.

- Evaluation is in order of precedence.

- Operators at same level are evaluated left to right for most operators.
# Precedence Rules

<table>
<thead>
<tr>
<th>Order</th>
<th>Group</th>
<th>Operator</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>High</td>
<td>Subexpression</td>
<td>()</td>
<td>Starting with innermost ()</td>
</tr>
<tr>
<td></td>
<td>Unary operators</td>
<td>-, +</td>
<td>Left to right.</td>
</tr>
<tr>
<td></td>
<td>Multiplicative operators</td>
<td>*, /, %</td>
<td>Left to right.</td>
</tr>
<tr>
<td>Low</td>
<td>Additive operators</td>
<td>+, -</td>
<td>Left to right.</td>
</tr>
</tbody>
</table>
Precedence Examples
Precedence Examples

\[ x + 4*y - \frac{x}{z} + \frac{2}{x} = ? \]
Precedence Examples

\[ x + \frac{4\times y}{1} - \frac{x}{z} + \frac{2}{x} = ? \]
Precedence Examples

\[ \frac{x}{1} + 4*y - \frac{x}{z} + \frac{2}{x} = ? \]
Precedence Examples

\[ x + \frac{4 \times y}{1} - \frac{x}{z} + \frac{2}{x} = ? \]
Precedence Examples

\[
x + \frac{4*y}{1} - \frac{x/z}{2} + \frac{2}{x} = ?
\]
Precedence Examples

\[
x + \frac{4 \times y}{1} - \frac{x}{z} + \frac{2}{x} = ?
\]
Precedence Examples

\[ \frac{4}{1} + \frac{4\times y}{2} - \frac{x / z}{3} + \frac{2 / x}{3} = ? \]
Precedence Examples

\[
x + \frac{4 \times y}{1} - \frac{x}{z} + \frac{2}{x} = ?
\]

\[
x + \left(4 \times y\right) - \left(\frac{x}{z}\right) + \left(\frac{2}{x}\right)
\]
Precedence Examples

\[
\begin{align*}
x & + \frac{4*y}{1} - \frac{x/z}{2} + \frac{2/x}{3} = ? \\
x & + (4*y) - (x/z) + (2/x)
\end{align*}
\]

\[
(x + y \times (4 - x) / z + 2 / -x) = ?
\]
Precedence Examples

\[ x + \frac{4*y}{1} - \frac{x/z}{2} + \frac{2}{x} = ? \]
\[ x + (4*y) - (x/z) + \frac{2}{x} \]

\[ (x + y \times \frac{4 - x}{z} + 2 / -x) = ? \]
Precedence Examples

\[
x + \frac{4*y}{1} - \frac{x/z}{2} + \frac{2/x}{3} = ?
\]
\[
x + (4*y) - (x/z) + (2/x)
\]

\[
(x + y) * \frac{(4 - x)}{z} + \frac{2}{-x} = ?
\]
Precedence Examples

\[
x + \frac{4 \times y}{1} - \frac{x}{z} + \frac{2}{x} = \?
\]

\[
x + (4 \times y) - \left( \frac{x}{z} \right) + \left( \frac{2}{x} \right)
\]

\[
(x + y) \times \left( \frac{4}{3} - \frac{\times}{1} \right) / z + \frac{2}{-x} = \?
\]
Precedence Examples

\[ x + \frac{4 \times y}{1} - \frac{x}{z} + \frac{2}{x} = ? \]
\[ x + (4 \times y) - \left( \frac{x}{z} \right) + \left( \frac{2}{x} \right) \]

\[ (x + y) \times (4 - x) \div z + \frac{2}{-x} = ? \]
Precedence Examples

\[
x + \frac{4 \times y}{1} - \frac{x}{z} + \frac{2}{x} = ?
\]

\[
x + (4 \times y) - \left(\frac{x}{z}\right) + \left(\frac{2}{x}\right)
\]

\[
(x + y \times (4 - x) \div \frac{z}{1} + 2 \div \frac{-x}{2}) = ?
\]
Precedence Examples

\[ x + \frac{4\times y}{1} - \frac{x}{z} + \frac{2}{x} = ? \]

\[ x + (4\times y) - \left( \frac{x}{z} \right) + \left( \frac{2}{x} \right) \]

\[ (x + \frac{y}{3} \times \frac{(4 - x)}{1}) \div \frac{z}{2} + \frac{2}{-x} = ? \]
Precedence Examples

\[ x + \frac{4\cdot y}{1} - \frac{x}{z} + \frac{2}{x} = ? \]
\[ x + (4\cdot y) - \frac{x}{z} + \frac{2}{x} \]

\[ (x + \frac{y}{3} \cdot (\frac{4}{1} - \frac{x}{z} + \frac{2}{-x})) = ? \]
Precedence Examples

\[
x + \frac{4 \times y}{1} - \frac{x}{z} + \frac{2}{x} = ?
\]

\[
x + (4 \times y) - \left(\frac{x}{z}\right) + \left(\frac{2}{x}\right)
\]

\[
\frac{6}{x} + y \times \frac{4 - x}{3} \div \frac{z}{1} + \frac{2}{x \div -x} = ?
\]

\[
(x + ((y \times (4-x)) \div z) + \left(\frac{2}{-x}\right))
\]
Precedence Examples

\[
x \begin{array}{c}
\text{1} \\
+ \\
\text{4} \times y \\
\text{5} \\
- \\
\text{x/z} \\
\text{6} \\
+ \\
2/x \\
\end{array} = ?
\]

\[
x + (4 \times y) - (x/z) + (2/x)
\]

To be safe, use parentheses!
Announcements

- Assignment 1 due tonight @ 10:00pm
- Assignment 2 will be released tomorrow.
- Exam 1 is on Feb 18th 6:30pm -- 7:30pm
  MTHW 210
Clarification on Division

- Integer division yields a truncated integer answer.
  - $14/3 = 4$
  - $14/-3 = -4$
  - $-14/3 = -4$
  - $-14/-3 = 4$
Clarification on Modulo

- Modulo’s sign matches the dividend’s sign.
- Modulo satisfies the following:
  - \( a = \left( \frac{a}{b} \right) \times b + (a \mod b) \)

For example:
- \( 14 \mod 3 = 2 \)
- \( 14 \mod -3 = 2 \)
- \( -14 \mod 3 = -2 \)
- \( -14 \mod -3 = -2 \)
Expression Types

- What is the data type of $i + j$;
- Depends upon the types of $i$ and $j$.
- If they are both
  - `int` then the result is also an `int`
  - `double` then the result is also a `double`
  - `long ... long`
  - Etc.
- Similarly for the other operators: -, *, ...
Type Casting

- If $x$ is a **float** and $y$ is an **int**, what is the data type of

  $$x \times y$$

  The answer is **float**.

- The above expression is called a **mixed expression**.

- Operands in mixed expressions are converted to a common type based on **promotion rules**.

- All are converted to the type with the highest precision in the expression.

- The entire expression is of this type too.
Implicit Type Casting

- Consider the following expression:
  \[
  \text{double } x = 3 + 5;
  \]

  - The result of 3 + 5 is of type \text{int}. However, since the variable \text{x} is \text{double}, the value 8 (type \text{int}) is promoted to 8.0 (type \text{double}) before being assigned to \text{x}.

- Notice that it is a promotion. Demotion is not allowed.

  \[
  \text{int } x = 3.5;
  \]
Implicit Type Casting

Consider the following expression:

```java
double x = 3 + 5;
```

The result of 3 + 5 is of type `int`. However, since the variable `x` is `double`, the value 8 (type `int`) is promoted to 8.0 (type `double`) before being assigned to `x`.

```java
byte → short → int → long → float → double
```

Notice that it is a promotion. Demotion is not allowed.

```java
int x = 3.5;
```
Implicit Type Casting

Consider the following expression:

```java
double x = 3 + 5;
```

The result of 3 + 5 is of type `int`. However, since the variable `x` is `double`, the value 8 (type `int`) is promoted to 8.0 (type `double`) before being assigned to `x`.

- `byte` → `short` → `int` → `long` → `float` → `double`

Notice that it is a promotion. Demotion is not allowed.

```java
int x = 3.5;
```

A higher precision value cannot be assigned to a lower precision variable.
Explicit Type Casting

Instead of relying on the promotion rules, we can make an explicit type cast:

( <data type> ) <expression>

Example

(float) x / 3

(int) (x / y * 3.0)

NOTE: Only the type of the return values is changed -- not the data itself.
Explicit Type Casting

- Instead of relying on the promotion rules, we can make an explicit type cast:
  \[( \text{<data type>} ) \text{<expression>} \]

- Example
  \[(\text{float}) \ x / 3 \]
  \[(\text{int}) (x / y * 3.0)\]

- **NOTE**: Only the type of the return values is changed -- not the data itself.
Explicit Type Casting

■ Instead of relying on the promotion rules, we can make an explicit type cast:
  \[(\text{<data type}> ) \text{<expression>}\]

■ Example
  \[(\text{float}) \ x \ / \ 3\]

  \[(\text{int}) \ (x \ / \ y \ * \ 3.0)\]

■ NOTE: Only the type of the return values is changed -- not the data itself.

Type cast \(x\) to \text{float} and then divide it by 3.

Type cast the result of the expression \(x \ / \ y \ * \ 3.0\) to \text{int}. 
Explicit demotion

- Promotion is automatically done whenever necessary.
- Demotion is not automatic, but can be forced:
  ```
  int x;
  double y;
  y = 3.5;
  x = (int)y;
  ```
- Assigning double (or float) to integer types results in \textit{truncation} (not rounding).
We can change the value of a variable. If we want the value to remain the same, we use a *constant*.

```java
final double PI = 3.14159;
final int MONTH_IN_YEAR = 12;
final short FARADAY_CONSTANT = 23060;
final float CARBON_MASS = 12.034F;
```
We can change the value of a variable. If we want the value to remain the same, we use a **constant**.

```java
final double PI = 3.14159;
final int MONTH_IN_YEAR = 12;
final short FARADAY_CONSTANT = 23060;
final float CARBON_MASS = 12.034F;
```

The reserved word `final` is used to declare constants.

These are constants, also called **named constants**.

These are called **literal constants**.
Why use Constants?

- **Consistent value**
  - No errors due to mistyping.
- **Easy to manage**
  - If we need to change the precision of PI, then we change it only once in the program.
- **Programs are more readable.**
Shorthand operators

- Some assignments and operators are combined into one operator to ease programming.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Usage</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>+=</td>
<td>a+=b;</td>
<td>a=a+b;</td>
</tr>
<tr>
<td>-=</td>
<td>a-=b;</td>
<td>a=a-b;</td>
</tr>
<tr>
<td>*=</td>
<td>a*=b;</td>
<td>a=a*b;</td>
</tr>
<tr>
<td>/=</td>
<td>a/=b;</td>
<td>a=a/b;</td>
</tr>
<tr>
<td>%=</td>
<td>a%≡b;</td>
<td>a=a%b;</td>
</tr>
</tbody>
</table>
Remember Methods

- Methods are used to perform sub-tasks.

To execute a method on an object:
  - myObject.methodName()
  - fullName.length()

To execute special class methods without an object:
  - ClassName.methodName();

Some methods need arguments
  - fullName.substring(2,3);
  - the method specifies the order and types
Some methods return values.
- `fullName.substring(2,3)`

The method specifies the type of the return value.
- `String substring (start:int, end: int)`

We can save the return value in an appropriate variable to use it later.

Methods can be composed
- `fullName.substring(getStart(), getEnd());`
Type Mismatch

- Suppose we want to input an age. Will this work?

```java
int age;
age = JOptionPane.showInputDialog(null, "Enter your age");
```
Type Mismatch

- Suppose we want to input an age. Will this work?

```java
int age;

age = JOptionPane.showInputDialog(null, "Enter your age");
```

- No. A string value cannot be assigned directly to an int variable.
Type Conversion

- **Wrapper classes** are used to perform necessary type conversions, such as converting a String object to a numerical value.
Wrapper classes are used to perform necessary type conversions, such as converting a String object to a numerical value.

```java
int age;
String inputStr;

inputStr = JOptionPane.showInputDialog(null, "Enter your age");

age = Integer.parseInt(inputStr);
```
## Other Conversion Methods

<table>
<thead>
<tr>
<th>Class</th>
<th>Method</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>parseInt</td>
<td>Integer.parseInt(&quot;25&quot;) → 25 \nInteger.parseInt(&quot;25.3&quot;) → error</td>
</tr>
<tr>
<td>Long</td>
<td>parseLong</td>
<td>Long.parseLong(&quot;25&quot;) → 25L \nLong.parseLong(&quot;25.3&quot;) → error</td>
</tr>
<tr>
<td>Float</td>
<td>parseFloat</td>
<td>Float.parseFloat(&quot;25.3&quot;) → 25.3F \nFloat.parseFloat(&quot;ab3&quot;) → error</td>
</tr>
<tr>
<td>Double</td>
<td>parseDouble</td>
<td>Double.parseDouble(&quot;25&quot;) → 25.0 \nDouble.parseDouble(&quot;ab3&quot;) → error</td>
</tr>
</tbody>
</table>
CAUTION: Imprecision

- It is not possible to exactly represent every possible float (double) number
  - Fixed number of bits
    - Float: 4 bytes -- 32 bits: $2^{32}$ (~1 billion) values
    - double: 8 bytes -- 64 bits: $2^{64}$ (~1 million trillion) values
  - Infinite numbers (e.g. between 1.0 and 2.0)!
- Floats and doubles may only store an approximation of the actual number!!!!
- Do not rely on exact values!
- Integers are stored precisely though!
Sample Code Fragment

```java
//code fragment to input radius and output
//area and circumference
double radius, area, circumference;

radiusStr = JOptionPane.showMessageDialog(null, "Enter radius: ");

radius = Double.parseDouble(radiusStr);

//compute area and circumference
area = PI * radius * radius;
circumference = 2.0 * PI * radius;

JOptionPane.showMessageDialog(null,
    "Given Radius: " + radius + "\n" +
    "Area: " + area + "\n" +
    "Circumference: " + circumference);
```
Overloaded Operator +

- The plus operator + can mean two different operations, depending on the context.
- `<val1> + <val2>` is an addition if both are numbers. If either one of them is a String, then it is a concatenation.
- Evaluation goes from left to right.
Overloaded Operator +

- The plus operator + can mean two different operations, depending on the context.
- `<val1> + <val2>` is an addition if both are numbers. If either one of them is a String, then it is a concatenation.
- Evaluation goes from left to right.

```java
 output = “test” + 1 + 2;
```
Overloaded Operator +

- The plus operator + can mean two different operations, depending on the context.
- `<val1> + <val2>` is an addition if both are numbers. If either one of them is a String, then it is a concatenation.
- Evaluation goes from left to right.

```
output = "test" + 1 + 2;
```
Overloaded Operator +

- The plus operator + can mean two different operations, depending on the context.
- `<val1> + <val2>` is an addition if both are numbers. If either one of them is a String, then it is a concatenation.
- **Evaluation goes from left to right.**

```
output = "test" + 1 + 2;
output = 1 + 2 + "test";
```
Overloaded Operator +

- The plus operator + can mean two different operations, depending on the context.
- `<val1> + <val2>` is an addition if both are numbers. If either one of them is a String, then it is a concatenation.
- Evaluation goes from left to right.

```java
output = "test" + 1 + 2;
```

```java
output = 1 + 2 + "test";
```
The DecimalFormat Class
The DecimalFormat Class

- Use a DecimalFormat object to format the numerical output.
The DecimalFormat Class

- Use a `DecimalFormat` object to format the numerical output.

```java
double num = 123.45789345;
DecimalFormat df = new DecimalFormat("0.000");
//three decimal places
```
The DecimalFormat Class

- Use a `DecimalFormat` object to format the numerical output.

```java
double num = 123.45789345;
DecimalFormat df = new DecimalFormat("0.000");
//three decimal places
System.out.print(num);
System.out.print(df.format(num));
```
The DecimalFormat Class

- Use a DecimalFormat object to format the numerical output.

```java
double num = 123.45789345;

DecimalFormat df = new DecimalFormat("0.000");  //three decimal places

System.out.print(num);
123.45789345

System.out.print(df.format(num));
```

```java
123.45789345
```
The DecimalFormat Class

- Use a **DecimalFormat** object to format the numerical output.

```java
double num = 123.45789345;
DecimalFormat df = new DecimalFormat("0.000");  
//three decimal places

System.out.print(num);
// 123.45789345
System.out.print(df.format(num));
// 123.458
```
The Math class

The **Math** class in the **java.lang** package contains class methods for commonly used mathematical functions.

```java
double num, x, y;
x = ...;
y = ...;
num = Math.sqrt(Math.max(x, y) + 12.4);
```
The Math class

The **Math** class in the **java.lang** package contains class methods for commonly used mathematical functions.

```java
double num, x, y;
x = ...;
y = ...;
num = Math.sqrt(Math.max(x, y) + 12.4);
```

- Table 3.7 in the textbook contains a list of class methods defined in the **Math** class.
Some Math Class Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Input type</th>
<th>Output type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>exp(a)</td>
<td>double</td>
<td>double</td>
<td>Return $e$ raised to power $a$.</td>
</tr>
<tr>
<td>log(a)</td>
<td>double</td>
<td>double</td>
<td>Return natural log of $a$.</td>
</tr>
<tr>
<td>floor(a)</td>
<td>double</td>
<td>double</td>
<td>Return largest whole number smaller than $a$.</td>
</tr>
<tr>
<td>max(a,b)</td>
<td>int</td>
<td>int</td>
<td>Return larger of $a$ or $b$.</td>
</tr>
<tr>
<td></td>
<td>double</td>
<td>double</td>
<td></td>
</tr>
<tr>
<td>pow(a,b)</td>
<td>double</td>
<td>double</td>
<td>Return $a$ raised to power $b$.</td>
</tr>
<tr>
<td>sqrt(a)</td>
<td>double</td>
<td>double</td>
<td>Return square root of $a$.</td>
</tr>
<tr>
<td>sin(a)</td>
<td>double</td>
<td>double</td>
<td>Return sine of $a$ (in radians).</td>
</tr>
</tbody>
</table>

Table 3.7 in the textbook contains a list of class methods defined in the Math class.
Computing the Height of a Pole

\[ h = \frac{d \sin \alpha \sin \beta}{\sqrt{\sin(\alpha + \beta) \sin(\alpha - \beta)}} \]

```
alphaRad = Math.toRadians(alpha);
betaRad = Math.toRadians(beta);

height = ( distance * Math.sin(alphaRad) * Math.sin(betaRad) )
         / Math.sqrt( Math.sin(alphaRad + betaRad) *
                        Math.sin(alphaRad - betaRad) );
```
The GregorianCalendar Class

- Use a `GregorianCalendar` object to manipulate calendar information

```java
GregorianCalendar today, independenceDay;

today    = new GregorianCalendar();

independenceDay
         = new GregorianCalendar(1776, 6, 4);
  // month 6 means July; 0 means January
```
Retrieving Calendar Information

This table shows the class constants for retrieving different pieces of calendar information from `Date`.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>YEAR</td>
<td>The year portion of the calendar date</td>
</tr>
<tr>
<td>MONTH</td>
<td>The month portion of the calendar date</td>
</tr>
<tr>
<td>DATE</td>
<td>The day of the month</td>
</tr>
<tr>
<td>DAY_OF_MONTH</td>
<td>Same as DATE</td>
</tr>
<tr>
<td>DAY_OF_YEAR</td>
<td>The day number within the year</td>
</tr>
<tr>
<td>DAY_OF_MONTH</td>
<td>The day number within the month</td>
</tr>
<tr>
<td>DAY_OF_WEEK</td>
<td>The day number within the week (Sun --1, Mon -- 2, etc.)</td>
</tr>
<tr>
<td>WEEK_OF_YEAR</td>
<td>The week number within the year</td>
</tr>
<tr>
<td>WEEK_OF_MONTH</td>
<td>The week number within the month</td>
</tr>
<tr>
<td>AM_PM</td>
<td>The indicator for AM or PM (AM -- 0, PM -- 1)</td>
</tr>
<tr>
<td>HOUR</td>
<td>The hour in the 12-hour notation</td>
</tr>
<tr>
<td>HOUR_OF_DAY</td>
<td>The hour in 24-hour notation</td>
</tr>
<tr>
<td>MINUTE</td>
<td>The minute within the hour</td>
</tr>
</tbody>
</table>
GregorianCalendar cal = new GregorianCalendar();
   //Assume today is Nov 9, 2003

System.out.print("Today is ") +
         (cal.get(Calendar.MONTH)+1) + "/" +
         cal.get(Calendar.DATE) + "/" +
         cal.get(Calendar.YEAR));
Sample Calendar Retrieval

GregorianCalendar cal = new GregorianCalendar();
//Assume today is Nov 9, 2003

System.out.println("Today is " + 
(cal.get(Calendar.MONTH)+1) + "/" + 
cal.get(Calendar.DATE) + "/" + 
cal.get(Calendar.YEAR));

Output

Today is 11/9/2003
Problem Statement

Problem statement:

Write a loan calculator program that computes both monthly and total payments for a given loan amount, annual interest rate, and loan period.
Overall Plan

Tasks:

- Get three input values: `loanAmount`, `interestRate`, and `loanPeriod`.
- Compute the monthly and total payments.
- Output the results.
Required Classes

LoanCalculator

JOptionPane

System.out : PrintStream

Math
We will develop this program in four steps:

1. Start with code to accept three input values.
2. Add code to output the results.
3. Add code to compute the monthly and total payments.
4. Update or modify code and tie up any loose ends.
Step 1 Design

- Call the `showInputDialog` method to accept three input values:
  - loan amount,
  - annual interest rate,
  - loan period.

- Data types are
Step 1 Design

- Call the `showInputDialog` method to accept three input values:
  - loan amount,
  - annual interest rate,
  - loan period.

- Data types are

<table>
<thead>
<tr>
<th>Input</th>
<th>Format</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>loan amount</td>
<td>dollars and cents</td>
<td>double</td>
</tr>
<tr>
<td>annual interest rate</td>
<td>in percent (e.g., 12.5)</td>
<td>double</td>
</tr>
<tr>
<td>loan period</td>
<td>in years</td>
<td>int</td>
</tr>
</tbody>
</table>
Step 1 Code

Program source file is too big to list here. From now on, we ask you to view the source files using your Java IDE.

**Directory:** Chapter3/Step1

**Source File:** Ch3LoanCalculator.java
Step 1 Test

- In the testing phase, we run the program multiple times and verify that:
  - we can enter three input values
  - we see the entered values echo-printed correctly on the standard output window
Step 2 Design

- We will consider the display format for out.
- Two possibilities are (among many others)

<table>
<thead>
<tr>
<th>Only the computed values (and their labels) are shown</th>
<th>Monthly payment: $143.47</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total payment: $17216.50</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Both the input and computed values (and their labels) are shown</th>
<th>For</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loan Amount: $10000.00</td>
<td></td>
</tr>
<tr>
<td>Annual Interest Rate: 12.0%</td>
<td></td>
</tr>
<tr>
<td>Loan Period (years): 10</td>
<td></td>
</tr>
<tr>
<td>Monthly payment is $143.47</td>
<td></td>
</tr>
<tr>
<td>TOTAL payment is $17216.50</td>
<td></td>
</tr>
</tbody>
</table>
Step 2 Code

Directory: Chapter3/Step2

Source File: Ch3LoanCalculator.java
Step 2 Test

- We run the program numerous times with different types of input values and check the output display format.
- Adjust the formatting as appropriate
Step 3 Design

- The formula to compute the geometric progression is the one we can use to compute the monthly payment.
- The formula requires the loan period in months and interest rate as monthly interest rate.
- So we must convert the annual interest rate (input value) to a monthly interest rate (per the formula), and the loan period to the number of monthly payments.
Step 3 Code

Directory: Chapter3/Step3

Source File: Ch3LoanCalculator.java
Step 3 Test

- We run the program numerous times with different types of input values and check the results.

<table>
<thead>
<tr>
<th>Loan Amount</th>
<th>Annual Interest Rate</th>
<th>Loan Period (in years)</th>
<th>Monthly Payment</th>
<th>Total Payment</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000</td>
<td>10</td>
<td>10</td>
<td>132.151</td>
<td>15858.088</td>
</tr>
<tr>
<td>15000</td>
<td>7</td>
<td>15</td>
<td>134.824</td>
<td>24268.363</td>
</tr>
<tr>
<td>10000</td>
<td>12</td>
<td>10</td>
<td>143.471</td>
<td>17216.514</td>
</tr>
<tr>
<td>0</td>
<td>10</td>
<td>5</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>30</td>
<td>8.5</td>
<td>50</td>
<td>0.216</td>
<td>129.373</td>
</tr>
</tbody>
</table>

(Shown up to three decimal places only)
Step 4: Finalize

- We will add a program description
- We will format the monthly and total payments to two decimal places using DecimalFormat.

Directory: Chapter3/Step4

Source File: Ch3LoanCalculator.java