

Object-Oriented Programming I

Prof. Elmira Adeeb

Variables

Slides by Magdin Stoica

Updates by Georg Feil/Elmira Adeeb

Learning Outcomes

1. Identify the two main elements that are defined in a class
2. Characterize a variable, its purpose and properties
3. Define the role and characteristics of variable types
4. Categorize types used in the Java programming language
5. Identify naming conventions used for naming variables
6. Contrast primitive (built-in) types with user-defined types
7. Characterize the role of “literals” in a programming language
8. Categorize literals in the Java programming language by type
9. Define the role and composition of expressions
10. Categorize operators
11. Create simple programs that define primitive and object variable of different types, combine variables in expressions and print values.

Reading Assignments

- ❑ Introduction to Java Programming (required)
 - Chapter 2: Elementary Programming
 - Sections 2.1 to 2.12 (except 2.7)
 - Section 2.16
 - Sections 2.17 to 2.18 (except 2.17.3)
 - **Note:** Section 2.3 (reading input) will be covered in class **next week**
 - Chapter 3, Section 3.2
- ❑ Head First Java (recommended)
 - Chapter 3: Primitives and References: Know Your Variables
 - From the start of the chapter up to and including section “An object reference is just another variable value”



What are programs made of?

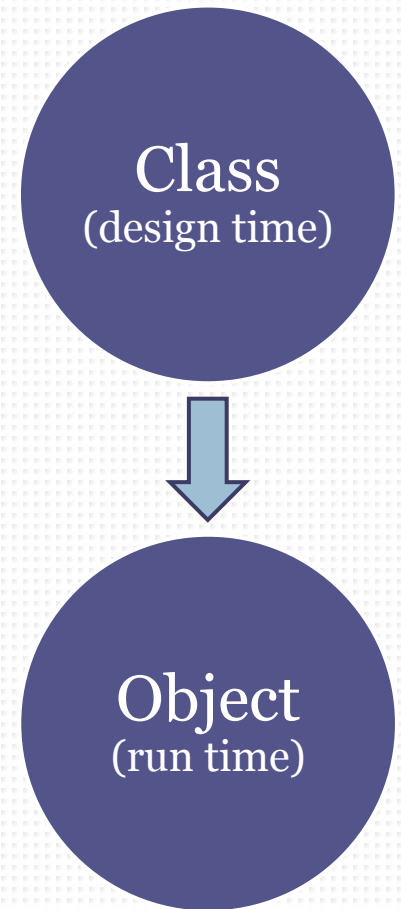
Objects

What defines objects?

Classes

Classes define what objects look like

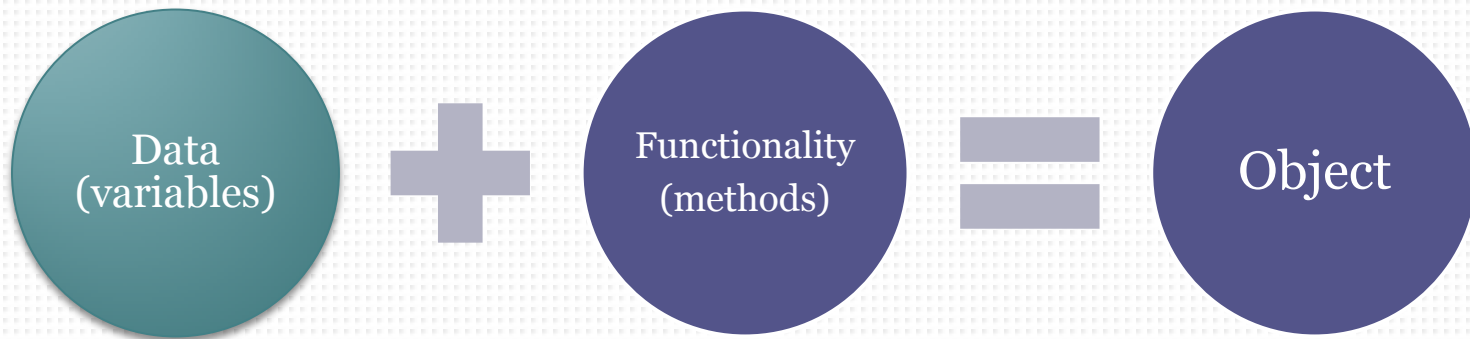
- ❑ **Classes** are things you implement at design time
 - Using the **class** keyword
 - All Java code consists of classes
 - All Java code is “part of” some class
 - Big programs can have many classes
 - Think of a class as a recipe to create objects
- ❑ **Objects** are created from classes at run time
 - Each object’s properties are defined by the class it was created from
 - There can be many objects made from one class
 - You can bake many cakes from one recipe



What are **objects** made of?

What's in a class / object?

- ❑ **Information** or **data** (what the objects will be made of) and **functionality** (what the object will be able to do)
- ❑ The elements of a class that contain information (data) are called **variables**.
- ❑ The elements of a class that execute commands to implement functionality are called **methods**.



Variables

- ❑ Objects contain data, information
 - A circle object will hold the dimension of the radius
 - A rectangle object will hold the width and the length of the rectangle
 - A bank account object will hold the balance in the account
 - A player object will hold the name of the player and the guess the player makes
- ❑ The elements of a program that **remember** information (data) are called **variables**
- ❑ A variable has 4 properties:
 - **Visibility** (or scope)
 - **Data type**
 - **Name**
 - **Value**

What are objects **made of**?

Objects are made of
things,
Variables

(and methods which we'll examine later)

Properties of Variables: Visibility

- The **visibility** of a variable determines which parts of the program can access (“see”) and therefore work with the variable
 - For some variables it must be **explicitly specified**
 - For some variables the visibility (or scope) is **implicitly determined** through the rules of the language
- Keywords for visibility: `public`, `private`
- Since in some cases visibility is implied, we don’t always see the `public/private` keywords

Properties of Variables: Data Type

- The **data type** of a variable determines the type of information it can remember. There are two categories of types:
 - **Primitive types** also known as **built-in types** which are types predefined by the programming language itself (numbers, characters etc.)
 - **User-defined types** are data types built from primitive types using classes
- Many user-defined types have already been created for you by people who wrote the **Java Library**
 - An example we've seen: `String`

Properties of Variables: Name

- The **name** of the variable is how programs refer to it
- You can choose almost any name you want for a variable
 - Some names are not allowed by the compiler
 - Some names are undesirable according to our own programming rules or **standards**, but the compiler doesn't complain about them

Properties of Variables: Value

- The variable's **value** is the actual information that is being remembered
- The kind of information (value) that can be stored in a variable is determined by its data type
 - whole numbers
 - floating point numbers
 - characters
 - strings
 - various kinds of objects

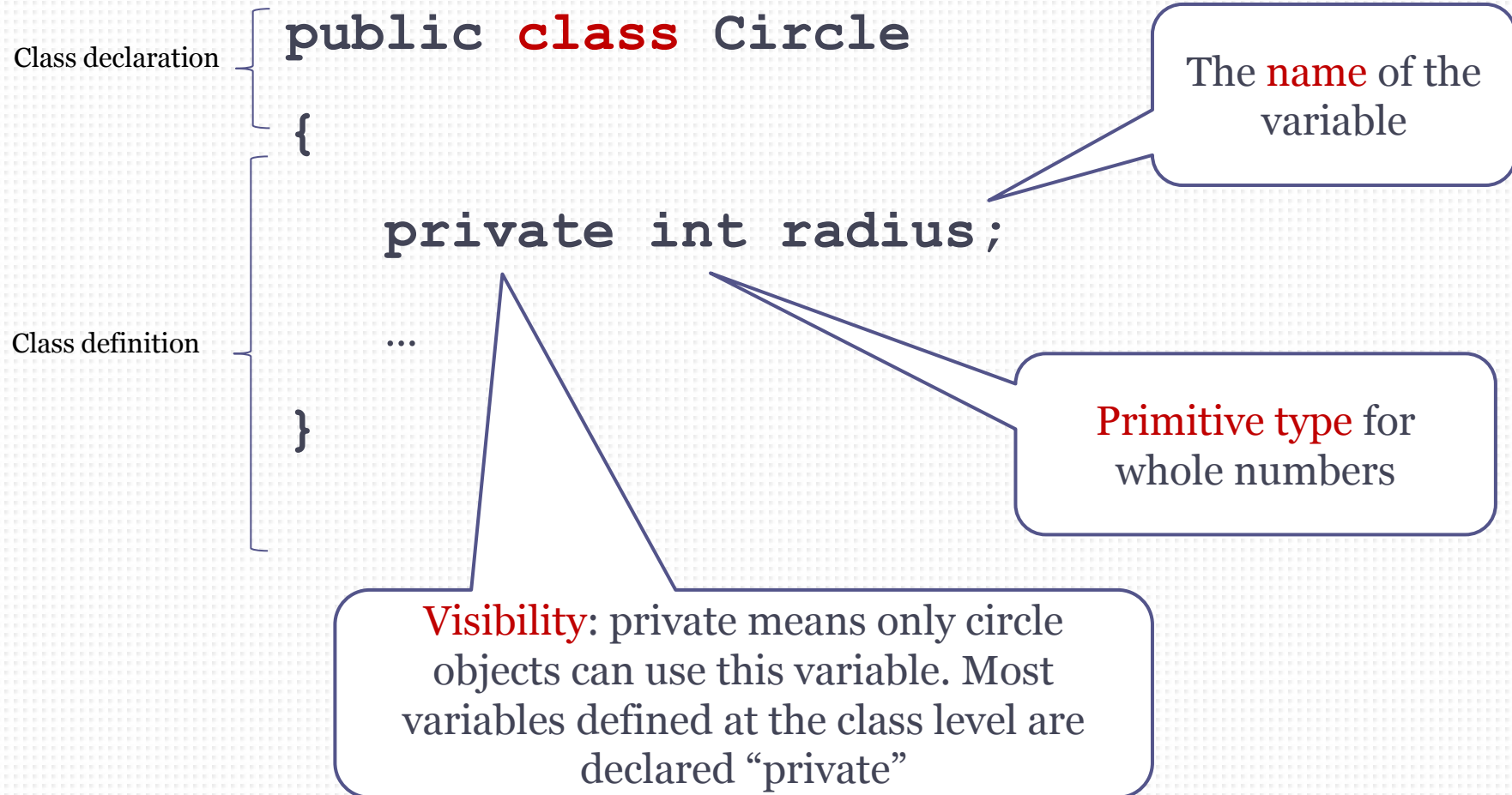
Declaring a variable (pseudocode)

Variable declaration



```
<visibility> <type> <name>;
```

Declaring a variable in a class



Data Types

- The **type** is one of the most important characteristics of a variable
 - It **constrains what values** can the variable remember. Can it remember whole numbers, real numbers, characters, true / false?
 - Compiler needs to know the type to know **how much memory** is needed to store the values the variable will hold
- **Primitive types** are defined in the programming language itself
 - **Numeric types** to remember numbers
 - **Character type** to remember characters text is made of
 - **Boolean type** to remember the values true and false which are of great importance in a program
- **User-defined types** are all other types defined using the primitive types by programmers
 - Can be pre-defined in class libraries or defined directly in programs.

Primitive Types: Numeric

- Numeric types are divided further into integral types and floating types
- **Integral types** used to remember whole numbers
 - byte : numbers are stored in 8 bits (-128 to 127)
 - short : numbers are stored in 16 bits (-32,768 to 32,767)
 - int : numbers are stored in 32 bits (-2,147,483,648 to 2,147,483,647)
 - long : numbers are stored in 64 bits (-2^{63} to $2^{63}-1$)
- **Floating-point types** used to remember real numbers
 - float : numbers are stored in 32 bits (about 7 significant digits)
 - double : numbers are stored in 64 bits (about 15 significant digits)
- See page 45 in the textbook for more details

Primitive Types: Character and Boolean

- Character type is used to remember a single character (letter, digit, symbol)
 - `char` - characters are stored in 16 bits (Unicode / UTF-16)
- Boolean type is used to remember a true/false or yes/no decision
 - `boolean` - may be stored in as little as 1 bit (implementation specific)
- See pages 62, 82 in the textbook for more details

How many **primitive types**
are there?

User-Defined Types

- Classes are user-defined types
 - Example: A variable of type Player can only hold player objects and nothing else
- User-defined types can be **predefined** by the class libraries that are provided by the runtime
 - The Java Runtime Environment (JRE) provides hundreds of predefined classes for us to use in our programs (e.g. System, PrintStream)
 - **Generally useful classes that are used in many programs**
- Every time we define a new class, we create (define) a new type
 - We define new classes because the runtime provides no predefined class for us to use
 - These classes represent things that are **specific to a program**
 - Example: Application, Program, Game, Player, Circle

Variable Names

- ❑ Always **start with a letter or underscore** `_`
 - Technically they can also start with **\$** but that is reserved for the compiler so **do not use it!**
- ❑ They can contain **letters, numbers** and underscores
 - Besides the initial underscore used for certain types of variables avoid using the underscore
- ❑ If **multiple words** are used, Capitalize every word but the first
 - `firstName, lastName, primeNumber, smartProgrammer`
- ❑ Cannot use keywords (like `class` and `public`) for any name
- ❑ Have to be **descriptive** to let the reader know what kind of information they will hold (*coding convention*)
 - `int i;` `//what does this variable remember?`
 - `int grade;` `//what does this variable remember?`

Declaring a **variable**

Use descriptive names
starting with lower-case
letter

<type> <name>;

**Variables are declared
using a
statement**

Visibility may be
needed before the type

Example: Declaring Variables

```
int sum;  
boolean isCorrect;  
byte grade;  
short age;  
double _radius;  
double area;  
float average;  
Circle circle1;  
Player larry;  
String _name;  
String reallyLongName;
```

```
int @cool;  . . .  
String #evenCooler;  
double ^huh;  
byte ^doubleHuh?;  
String does-not-work;  
byte 1Number;
```

Red examples
are incorrect.
Why?

```
int Sum;  
String $imTheCompiler;  
double canyoureadthisname;  
byte some_like_this_not_me;
```

Orange examples
will compile but
are not used by
convention

Variables **Class EggBasket**

LISTING 2.1 A Simple Java Program

```
public class EggBasket
{
    public static void main(String[] args)
    {
        int numberOfBaskets, eggsPerBasket, totalEggs;
        numberOfBaskets = 10;
        eggsPerBasket = 6;
        totalEggs = numberOfBaskets * eggsPerBasket;
        System.out.println("If you have");
        System.out.println(eggsPerBasket + " eggs per basket and");
        System.out.println(numberOfBaskets + " baskets, then");
        System.out.println("the total number of eggs is " + totalEggs);
    }
}
```

Variable
declarations

Assignment statement

Sample Screen Output

```
If you have
6 eggs per basket and
10 baskets, then
the total number of eggs is 60
```

Exercise 1: Practice Primitive Variables

- Using Dr. Java and the Interactions Pane
 - Define the variables in the previous slide
 - Use all the primitive types we have learned
 - Experiment with different names and see what the compiler has to say about them

Remembering values

- Variables remember values, data, information in a program
 - **Initializing a variable**: storing a value in a variable for the first time
 - **Assigning a value to a variable**: changing the value a variable “remembers”
- Values are stored (remembered) in variables using an **assignment operator**
- A statement that changes a variable’s value is called an **assignment statement**

<variable name> = <value>;

- The variable has to have been declared first
- The value has to match the variable’s data type

Variables are called variables
because their **value changes**

Declaring + Initializing in One Step (pseudocode)

- ▣ Variables can be declared AND initialized using one combined statement
 - You must remember that two statements are combined and therefore can be separated

Variable declaration

```
<visibility> <type> <name> = <value>;
```

Variable initialization

Value has to match
the type

`<type> <name> = value;`

One **combined** statement
can be used to
declare and initialize
a variable

Visibility may be
needed before the type

Declaring + Initializing in One Step

- Here is what a Java variable declaration and initialization looks like for a variable declared at the top of a class
 - This variable has visibility private

Variable declaration

```
private double pi = 3.1415926535;
```

Variable initialization

Declarations and Definitions

Declarations and Definitions

- We've learned that we can create variables and give them names
 - We must declare variables before we can use them
- In the same way, we must declare other kinds of things and give them names before we can use them
 - Methods
 - Classes
 - Packages

Together **declarations** and **definitions** help define **new** “**words**” and their meaning in a program.

Things that need to be declared

1. **Variables** remember information
2. **Methods** define groups of statements that execute
3. **Classes** encapsulate variables (information) and methods (statements that work with that information)
4. **Packages** are groups of classes

Example: “Hello World” (v3)

```
package sheridan;  
public class Program3  
{  
    public static void main(String[] args)  
    {  
        // A Java program starts with the first  
        // statement in main  
        String message = "Hello From Program 3";  
        System.out.println(message);  
    }  
}
```

Exercise 4: Breaking Down Hello World

- ❑ Analyze the version of Hello World shown in the previous slide
- ❑ Using the provided map break down the program into elements and identify:
 - Declarations: how many and what is being declared
 - Blocks of statements: how many and what do they contain
 - Statements: how many statements and what do they do
- ❑ There is one declaration in our program that's not shown in the map, see if you can find it!

Object variables

- ❑ Objects are stored in variables whose data type is a class
 - In general we call them **object variables**
 - Specifically we may call them “<name of the class> variables”
 - Player variables if their type is the Player class or circle variables if their type is the Circle class
 - Actually the variable stores a **reference** to where the object is in memory
- ❑ Classes are **types**
 - You can make variables out of any type
- ❑ Object variables are declared the same way as simple variables but they are initialized (“created”) using the **new** keyword

```
<class name> <variable name> = new <class name>();
```

Object variables

- For example, you can create a 'Circle' object like this

```
Circle bigRound = new Circle();
```

- Or you can do this (inside a method only!)

```
Circle bigRound;  
bigRound = new Circle();
```

Literals

Literals

- **Literals** are constant values that appear directly in a program
 - Literals have a data type just like variables: int, double etc.
 - A literal cannot change – it is the value itself
 - Examples: 3, 5.6, true, “John Wayne”, ‘a’
- **Whole number** literals:
 - By **default** have the type “**int**”: 3, 6, 12345, -456
 - If **suffixed** with the letter **L** or **l** they are treated as having the type “**long**”: 3**L**, 6**L**, 9876543210**L**, -456**l**
- **Floating point number** literals:
 - By default have the type “double”: 4.5, -3.4, 0.46, 12345.456
 - If suffixed with the letter F or f they are treated as having the type “float”: 4.5**f**, -3.4**F**, 0.46**f**, 12345.678**F**
 - Any number literal with suffix d/D (for double) or f/F (float) is treated as a real number: 3f, 5F, 6d, 9D


Literals (cont.)

- Text literals
 - Single **character literals** are enclosed in **single quotes**: 'a', 'c', 'x'
 - **Multi-character literals** are called strings and are enclosed in **double quotes**: "abc", "john", "This is a string!", "z", ""
- Boolean literals
 - `true` – a condition that is true (such as `5 < 10`)
 - `false` – a condition that is false (such as `5 > 10`)
 - `true` and `false` are keywords
- Object literals
 - `null` – means “no object” or “nothing”
 - More on “null” later (it is also a keyword)

Examples

```
int value;  
value = 5;  
value = 10;  
value = "John";  
boolean isCorrect;  
isCorrect = true;  
isCorrect = 0;  
char ch;  
ch = 'x';  
ch = "Z";  
byte smallValue;  
smallValue = 78;  
smallValue = 123455;
```

```
Circle c;  
c = new Circle();  
Circle c2 = new Circle();  
c = new Rectangle();  
Program p;  
p = null;  
p = new Program();  
p = new Circle();
```



Red examples
are incorrect.
Why?

Exercise 2: Initializing Variables

- ❑ Using Dr. Java and the Interactions Pane
 - Define and initialize variables to different values
 - Print them by typing their name or using `System.out.println(<variable name>)`
- ❑ Using Dr. Java create a class called VariableTester
 - In the main method enter the variable declarations and assignments in the left-hand column on slide 42
 - Note the compilation errors that occur when the value does not match the type... fix the “red” examples... try some experiments of your own

Expressions

- Expressions are a calculation involving **literals** (values) or **variables** combined using **operators**
 - Method calls are another way to obtain / calculate values but we'll learn this in the next lesson
- Examples
 - $5 + 10 - 42 + 209$
 - $10 * 10 * 3.14$
 - $\text{radius} * \text{radius} * 3.14$
 - $\text{percent} / 100$

Arithmetic Operators

- Java has all the operators we know from math

Addition	+
Subtraction	-
Multiplication	*
Division	/
Remainder	%

- Operators have the same precedence (order) as in mathematics: BODMAS
- Expressions can contain other expressions. Use parentheses to clarify and ensure precedence: $10 * (3 + 4)$

Math Formula....C Expression

Mathematical Formula

Java Expression

$$b^2 - 4ac$$

$$b*b - 4*a*c$$

$$a + b - c$$

$$a + b - c$$

$$\frac{a + b}{c + d}$$

$$(a + b) / (c + d)$$

$$xyz + a/b - c*d$$

$$(x*y*z) + (a/b) - (c*d)$$

$$a_x - (b + c)$$

$$a^* - (b + c)$$

What is the result of the following expressions?

a) $(1+8)/9*2-10$

b) $1+8/9*2-10$

Expressions Examples

```
double average = (2.5 + 5.5 + 8.1) / 3.0;

int quotient = 4 / 3;
quotient = 2 / 3;
int remainder = 2 % 3;

int value = 50;
value = 10 * remainder;
value = 10 * value; // how much is 'value'?
value = (5 + 5) * value; // what about now?
value = value + 2; // what about now?
System.out.println(value);

String name = "Barack";
name = name + " ";
name = name + "Obama";
System.out.println(name); // what is 'name'?
```

Exercise 3: Expressions

- Using Dr. Java and the Interactions Pane
 - Define and initialize variables to different **expressions** as on the previous slide
 - Print them by typing their name or using `System.out.println(<variable name>)`
- Using Dr. Java load the VariableTester program from Exercise 2
 - Initialize the different variables to different expressions
 - Compile after each change and note the compilation errors that occur when the value does not match the type
- Textbook exercises:
 - 1.5: Computing expressions
 - 1.6: Summation of a series (of fixed size)
 - 1.7: Approximating Pi

Mathematical Methods

Function	Returns
<code>Math.sqrt(x)</code>	square root
<code>Math.pow(x, y)</code>	power x^y
<code>Math.exp(x)</code>	e^x
<code>Math.log(x)</code>	natural log
<code>Math.sin(x)</code> , <code>Math.cos(x)</code> , <code>Math.tan(x)</code>	sine, cosine, tangent (x in radians)
<code>Math.round(x)</code>	closest integer to x
<code>Math.min(x, y)</code> , <code>Math.max(x, y)</code>	minimum, maximum

Complex Expressions

- What about larger math expressions?

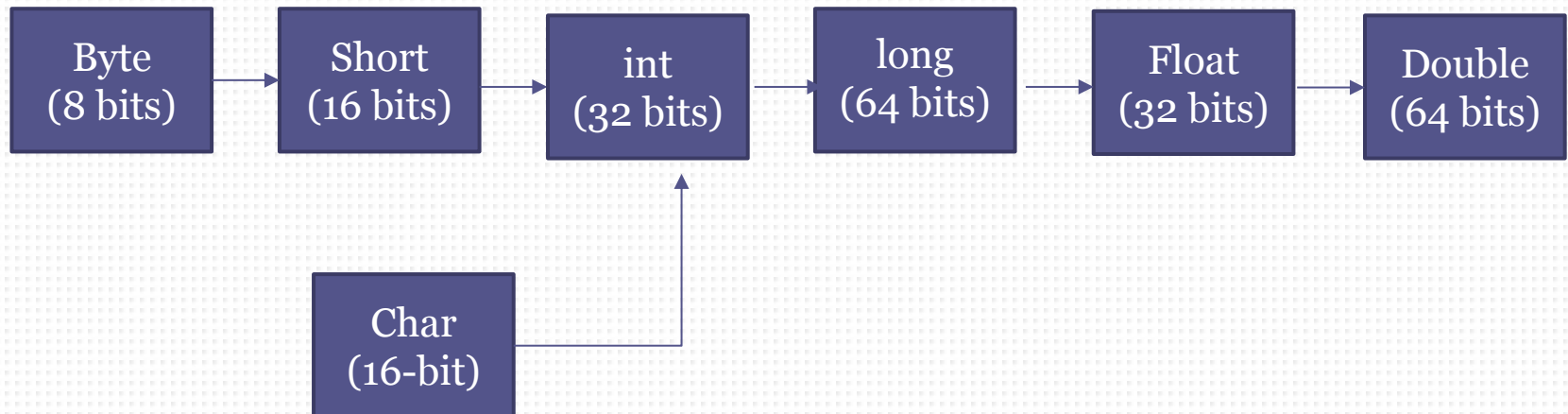
$$\sqrt{\frac{102 + 79 - 576.7}{3927}}$$

- We have to rewrite them in Java syntax...

```
double result = Math.sqrt((102 + 79 - 576.7) / 3927)
```

Type casting

- A *type cast* temporarily changes the value of a variable from the declared type to some other type.
- Order of implicit type casting



Example

Explicit typecasting

`double adouble=45; //the compiler will automatically type cast it to 45.00`

Implicit typecasting

`int anum=2, bnum=3;`

`double result1, result2, result3;`

`result1= (anum)/(bnum); → will be zero (0)`

`result2= (double)(anum)/(bnum); -> will be (.67)`

`result3= (double)(anum/bnum); -> what will be the result??`

Objects and Variables

- ❑ Objects remember information (attributes) in **field** variables
 - Fields are declared at the start of the class definition
 - We can use (access or change) fields in any method*
 - Each time we create an object using **new**, the new object has its own copy of all field variables
- ❑ How can we use the variables inside an object from “outside”?
 - How can we access them?
 - How can we modify them?

```
public class Circle {  
    private double radius = 10.0;  
    ...  
}
```

```
public class Person {  
    private String _firstName="Barack";  
    private String _lastName="Obama";  
  
    void printNameMethod() {  
        System.out.println(_firstName);  
    }  
    ...  
}
```

The . (dot) operator

- ❑ To access an element that is defined inside an object you must use the object variable followed by the . (dot) operator
 - Dot operator provides you access to the fields and methods defined inside the object
 - To be accessible from another class, the inner elements of the object must be visible outside of the object (public).
We don't normally do this for variables

- ❑ Don't let this confuse you:

```
Circle circle = new Circle();  
double radius = circle.radius;  
System.out.println(radius);
```

```
Circle circ = new Circle();
```

```
double rad = circ.radius ;
```

```
Person p = new Person();
```

```
p._firstName = "Barack";
```

```
p._lastName = "Obama"
```


Objects and Variables

- ❑ **circ.radius** identifies the radius variable inside the circle object “remembered” in variable ‘circ’
 - It’s value is 10;
- ❑ **p._firstName** identifies the String variable inside the Person object “remembered” in variable ‘p’
 - It’s value is “Barack”
- ❑ **p._lastName** identifies the String variable inside the circle object “remembered” in variable ‘p’
 - It’s value is “Obama”

```
Circle circ = new Circle();
```

```
double rad = circ.radius ;
```

```
Person p = new Person();
```

```
p._firstName = “Barack”;
```

```
p._lastName = “Obama”
```

Field variables

```
Class declaration { public class Person
                  {
Class definition {   private String _firstName;
                  private String _lastName;
                  ...
                  }
```

Variables declared directly in the class definition block (not in a method) are called **fields**

An **object** is made of its **fields**!



Variables Example: Extending Hello World

```
public class ProgramV
{
    // The mathematical constant Pi
    // Note: 'static' is needed here to use it from main method
    private static double pi;

    public static void main(String[] args)
    {
        // This writes a message to the screen
        System.out.println ( "Hello Pi Program" );
    }
}
```

Variables Example: Extending Hello World

```
public class ProgramV
{
    // The mathematical constant Pi
    // Note: 'static' is needed here to use it from main method
    private static double pi;

    public static void main(String[] args)
    {
        // This writes a message to the screen
        System.out.println ( "Hello Pi Program" );

        pi = 355.0 / 113.0;
        System.out.println ( "Pi is about " + pi );
    }
}
```

Variables Example: Extending Hello World

```
public class ProgramV
{
    // The mathematical constant Pi
    // Note: 'static' is needed here to use it from main method
    private static double pi;

    public static void main(String[] args)
    {
        // This writes a message to the screen
        System.out.println ( "Hello Pi Program" );

        pi = 355.0 / 113.0;
        System.out.println ( "Pi is about " + pi );

        // How much to multiply by (this is a local variable)
        double multiplier = 2;

        double twoPi = multiplier * pi;
        System.out.println ( "Two Pi is about " + twoPi );
    }
}
```

Recommended Exercises

- 1.5: Computing expressions
- 1.6: Summation of a series (of fixed size)
- 1.7: Approximating Pi
- 1.8: Area and perimeter of a circle
- 1.11: Population projection (*more advanced*)