Object Oriented Programming with Java

02 – Review of Basic Concepts

Important Remarks

- All Java code must be in a class saved in a file with the same name of the class!
  - ie-> Welcome.java contains Welcome class
  - All standard Java applications require main method to start

```
public static void main(String[] args){

}
```

Comments

- Comments start with: //
  - Comments ignored during program execution
  - Document and describe code
  - Provides code readability
- Traditional comments: /* ... */
  - /* This is a traditional comment. It can be split over many lines */

Identifiers

- Java identifier
  - Series of characters consisting of letters, digits, underscores (_) and dollar signs ($)
  - Does not begin with a digit, has no spaces
  - Examples: `Welcome1`, `$value`, `_value`, `button7`
    - `7button` is invalid
- Java is case sensitive (capitalization matters)
    - `a1` and `A1` are different
- Class names should begin with capital letters
- Saving files
  - File name must be class name with `.java` extension
  - `Welcome1.java`

Sabancı Üniversitesi

Data & Variables

- **Variables**
  - Modern day languages enable us to use symbolic names known as Variable which refer to the memory location where a particular value is to be stored
  - Variables enable the data to temporarily store within a program.
- **Data Types**
  - Defines the kind of value the variable can hold
  - For example, can this variable hold numbers? Can it hold text?

Note: Variables are not persistent. When you exit your program, the data is deleted. To create persistent data, you must store it to a file system.

Data & Variables

Primitive Data Types

- Integers
  - `byte` – 8 bits (values from -128 to +127)
  - `short` – 16 bits (-32768 to +32767)
  - `int` – 32 bits (-2 147 483 648 to 2 147 483 647) or (-2^{31} to $2^{31} - 1$)
  - `long` – 64 bits (even bigger numbers)
- Characters
  - `char` – 16 bits, represented in unicode, not ASCII!
- Floating point numbers
  - `float` – 4 bytes (-3.4×10^{38} to $+3.4 \times 10^{38}$)
  - `double` – 8 bytes (-1.7×10^{308} to 1.7×10^{308})
- Boolean data
  - `boolean`
    - can only have the value true or false

Data & Variables

- Every variable must have two things: a data type and a name.
- Use descriptive variable names that clarify the purpose of your code.

Example:

```
        area = PI * radius * radius;
        a = p * r * r;
```

- All variables must have declared type

```
 double salary;
   int vacationDays;
```

- Variables must be initialized before use

```
   salary = 54000.0;
```

- Can do both in one line

```
 char yesChar = 'y';
```

- Must begin with letter followed by letter/numbers (in Unicode)
- Do not use Turkish characters in variable names.

Data & Variables

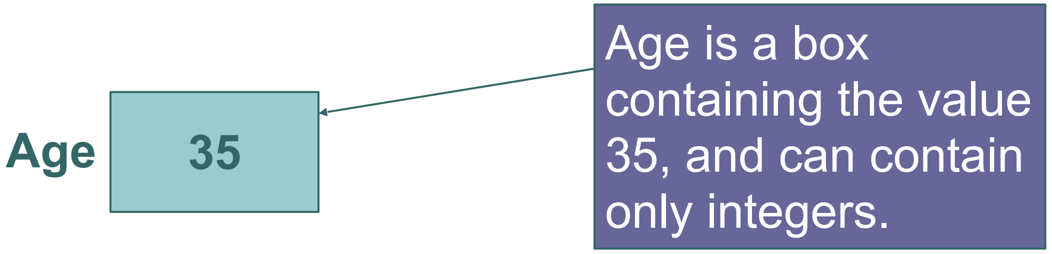
- Before use in program statements and expressions, variables must be declared.
- Declaring variables informs the computer how much memory to reserve for each variable, as well as initial values to store.
- Variable declaration format (simplified)

Examples:

```
int sumGrades, remainder, r2d2c3po;
double avg = 0.0, stdDev = 0.0;
char initial3 = 'T';
boolean completed = false;
```

Box analogy

```
int Age =35;
```


Age
35
Age is a box containing the value 35, and can contain only integers.

Data & Variables

Reference Data Types

- They are the objects we are going to use, and the reside on the *heap* side of memory
- Ex: System , Object
- They are initialized by using the *new* keyword
  - Ex: `Object o = new Object();`
- *String* is a reference type but acts like primitive types
  - Ex: `String message = "Welcome to Java!";`
- more on *reference types* and *String* object later…

Sabancı Üniversitesi

Operators

- Arithmetic

  + , - , * , / , % , ++ , --
- Logic

  && , || ..
- Assignment

  = , += , -= , etc..
- Relational

  < , <= , > , >= , == , !=

Operators - Arithmetic

- Arithmetic calculations used in most programs
  - Usage
    - `*` for multiplication
    - `/` for division
    - `%` for remainder
    - `+`, `-`
  - Integer division truncates remainder
    `7 / 5` evaluates to 1
  - Remainder operator % returns the remainder
    `7 % 5` evaluates to 2

Java operation	Arithmetic operator	Algebraic expression	Java expression
Addition	+	$f + 7$	`f + 7`
Subtraction	–	$p - c$	`p - c`
Multiplication	*	bm	`b * m`
Division	/	x / y or $\frac{x}{y}$ or $x \div y$	`x / y`

Sabancı Üniversitesi

Operators - Arithmetic

Operator(s)	Operation(s)	Order of evaluation (precedence)
* / %	Multiplication Division Remainder	Evaluated first. If there are several operators of this type, they are evaluated from left to right.
+ -	Addition Subtraction	Evaluated next. If there are several operators of this type, they are evaluated from left to right.

- parentheses used for overriding operator precedence

Sabancı Üniversitesi

Operators - Arithmetic

- Shorthand Operators
  - For all arithmetic operators we can both make a calculation and assignment at the same time
  - Ex:

```
int x = x + 7;
```

$\xrightarrow{\text{same as}}$

```
int x += 7;
```

```
int x = x / 3;
```

```
int x /= 3;
```

Operators - Arithmetic

- Shorthand Operators
  - ++ and -- operators are used for incrementing or decrementing a variable by 1

```
int x = 2;
int z = x++;
int y = x--;
```

Post increment/decrement

At the end all x, and z equals 2
y equals 3

```
int x = 2;
int z = ++x;
int y = --x;
```

Pre increment/decrement

At the end x = 2 , z = 3 and
y=2

Assignment Operator “=”

- Use assignment operator to point the references of primitive types to certain literals
- When a primitive variable is assigned to another, the contents of the right-hand variable are copied

```
int x = 12;
int y = x;
x++;
System.out.println(x);
System.out.println(y);
```

```
Output will be:
13
12
```

NOTE : This is not valid for reference types *(more later)*

String Concatenation

- Using the **`+`** operator with two **`String`**s concatenates them into a new **`String`**
- Using the **`+`** operator with a **`String`** and a value of another data type concatenates the **`String`** with a **`String`** representation of the other value
  - When the other value is an object, its **`toString`** method is called to generate its **`String`** representation

Some Common Escape Sequences

Escape sequence	Description
`\n`	Newline. Position the screen cursor at the beginning of the next line.
`\t`	Horizontal tab. Move the screen cursor to the next tab stop.
`\r`	Carriage return. Position the screen cursor at the beginning of the current line—do not advance to the next line. Any characters output after the carriage return overwrite the characters previously output on that line.
`\\`	Backslash. Used to print a backslash character.
`\"`	Double quote. Used to print a double-quote character. For example, `System.out.println( "\"in quotes\"" );` displays `"in quotes"`

String Formatting

- `String.format(format,values)` or `System.out.printf(format,values)` methods can format Strings:
  - Placeholders:

%f	**decimal**
%d	integer
%s	string

```
System.out.printf("There are %d apples
in %s's bag, for %.2f TL each.",12,"
John,3.45");
```

Sabancı Üniversitesi

Console Input

- Java uses `System.out` to refer to the standard output device and `System.in` for the input device
- As console input is not directly supported we use `Scanner` object to read input from `System.in` **(java.util.Scanner should be imported)**

```
System.out.println("Enter a double value:");
Scanner input = new Scanner(System.in);
double d = input.nextDouble();
```

Conditions & Loops

•When writing conditional statements, we get use of equality and relational operators:

Standard algebraic equality or relational operator	Java equality or relational operator	Sample Java condition	Meaning of Java condition
Equality operators			
=	==	x == y	x is equal to y
≠	!=	x != y	x is not equal to y
Relational operators			
>	>	x > y	x is greater than y
<	<	x < y	x is less than y
≥	>=	x >= y	x is greater than or equal to y
≤	<=	x <= y	x is less than or equal to y

Conditions & Loops

- ***<u>If</u>***

if *<u>condition</u>*
    <u>statement1</u>
End

İf *<u>name is Ahmet</u>*
    <u>Display ”Hello Ahmet”</u>

<u>Java syntax;</u>

```
if (name==“Ahmet”)
{
System.out.print(“Hello Ahmet”);
}
```

Braces defines the scope

Sabancı Üniversitesi

Conditions & Loops

○ ***If ..else***

if *condition* → if *name is Ahmet*

statement1 → *Display "Hello Ahmet"*

else → else

statement2 → Display "Hello Nobody"

Java syntax →

```
 if (name=="Ahmet")
   {
           System.out.print("Hello Ahmet");
   }
else
   {
           System.out.print("Hello Nobody");
   }
```

Conditions & Loops

- ***Nested if Statements***

```
 if (number < 10 && number >0)
{
   if (number > 5 )
         {
         System.out.print("The Number is between 5 and 10");
         }
    else
         {
          System.out.print("The Number is between 5 and 10");
         }
}
else
{
System.out.print("The number is not between 0 and 10");
}
```

Sabancı Üniversitesi

Conditions & Loops

- <u>Logical operators</u>
  - Allows for forming more complex conditions
  - Combines simple conditions
- <u>Java logical operators</u>
  - && (conditional AND)
  - | | (conditional OR)
  - & (boolean logical AND)
  - | (boolean logical inclusive OR)
  - ^ (boolean logical exclusive OR)
  - ! (logical NOT)

Conditions & Loops

- ***Switch***

```
String string =“”;
int i = 1;
switch ( i )
        {
                  case 1:
                  string = "foo";
                         break;
                  case 2:
                  string = "bar";
                          break;
                 default:
                  string = "";
                          break;
        }
System.out.println(string);
```

Condition variable

switch statements require the condition variable to be a **char, byte, short or int**

Condition check

Exit of the loop

if you forget to write “break”s the loop will check all cases and execute them

Conditions & Loops

- ***While***

```
int a = 0;
int i = 0;
While(i<100)
{
        a += i;
//      a = a + 1;
        i++;
}
System.out.println("Sum = "+ a);
```

Sabancı Üniversitesi

Conditions & Loops

- ***Do … While***

```
int a = 0;
int i = 0;
do
{
       a += i;
//     a = a + 1;
       i++;
} While(i<100);

System.out.println("Sum = "+ a);
```

Sabancı Üniversitesi

Conditions & Loops

- ***for***

for (*variable start point* ; *condition* ; *increment*)
{
  statements
}

```
int a = 0;
for( int i = 0 ; i < 100 ; i++ )
{
      a += i;
}
System.out.println("Sum = "+ a );
```

Conditions & Loops

- ***Break***

```
for( int i = 0 ; i < 10 ; i++ )
 {
 if( i == 3 )
      break;
   System.out.println(""+ i );
}
```

Breaks the loop
Will not continue after 3,

- ***Continue***

```
for( int i = 0 ; i < 10 ; i++ )
 {
 if( i == 5 )
      continue;
   System.out.println(""+ i );
}
```

if the condition is true loop will not execute following code in the loop scope & will continue for the next value

Branching Statements

- You can break an outer loop from an inner loop

```
outer:
for(int i=0;i<100;i++){
      for(int x=0;x<50;x++){
      if(x<30)
      System.out.println("i=" + i + " x=" + x);
      else
      break outer;
      }
}
```

Examples

- Create a program that informs people if they are old or young according to the input provided
- Create a gradebook calculator that takes a number of grades and calculates avarage, input=101 will break the program

Lab – Number Guess Game

- Computer will generate a number (1-10) randomly and in 5 steps at max users should be able to find the number
- Computer will give hints according to the input provided.

Arrays

- Variables used for store a single piece of information.If we need to store a series of *same kind of* data we use ***Arrays***
- Like variables, Arrays must be declared before use
- Like variables, Arrays must have a name & data type
- Unlike variables, size of Arrays must be declared before using

Declaring Arrays;

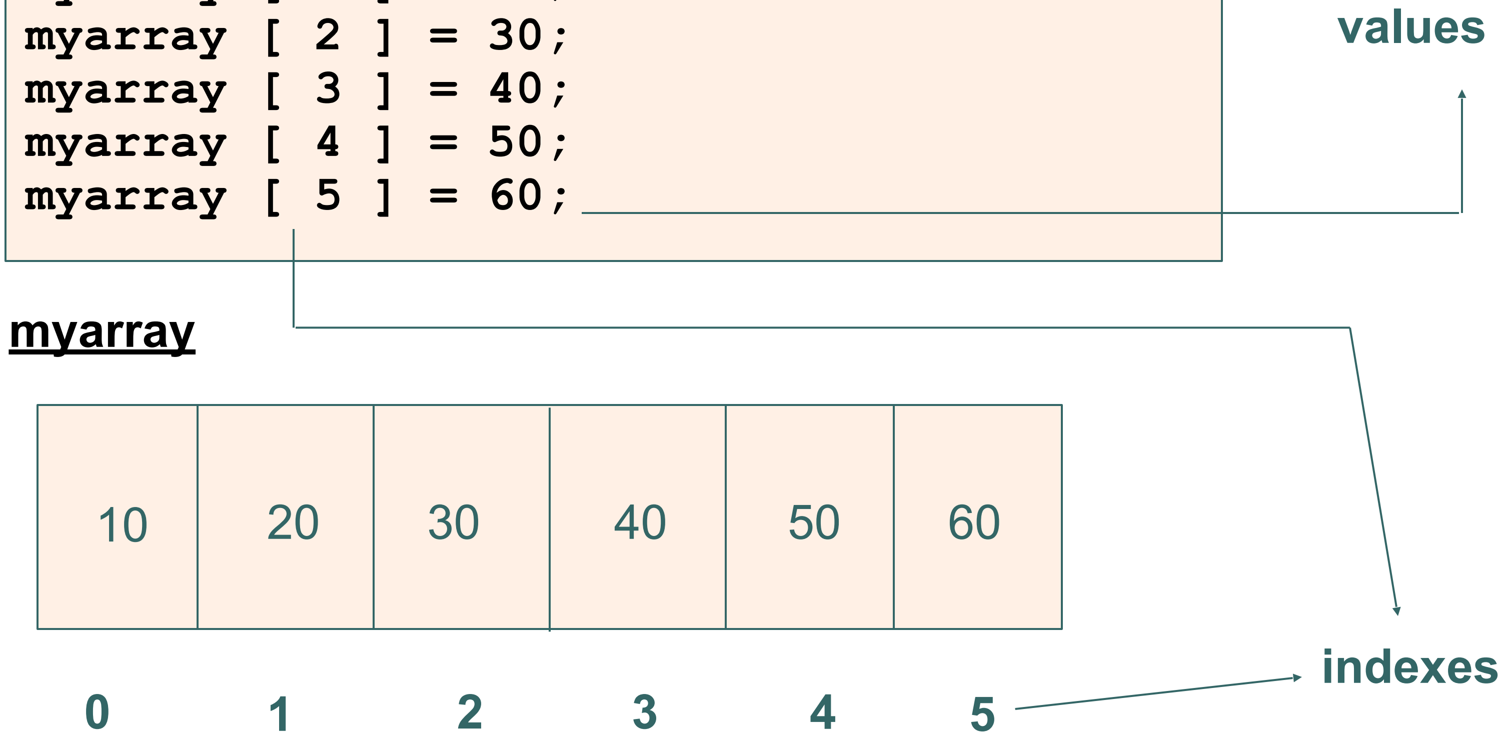
```
int myarray [ ] = new int [ 20 ];
char myStrings[] = new char [ 10 ];

//Or
int mynumbers[] = { 1 , 2 , 3 , 4 , 5 };
```

Arrays

- <u>Accessing Array Elements;</u>

```
int myarray [ ] = new int [ 6 ];
myarray [ 0 ] = 10;
myarray [ 1 ] = 20;
myarray [ 2 ] = 30;
myarray [ 3 ] = 40;
myarray [ 4 ] = 50;
myarray [ 5 ] = 60;
```



Arrays

- Enhanced `for` Statement
  - Iterates through elements of an array or a collection without using a counter
  - Syntax

```
for ( parameter : arrayName )
        statement
```

- Example:

```
int numbers[]  = {1,2,3,4,5,6,7,8,9};
for (int i : numbers) {
            System.out.println("rakam:  " + i);
}
```

Arrays

Multi Dimensional Arrays

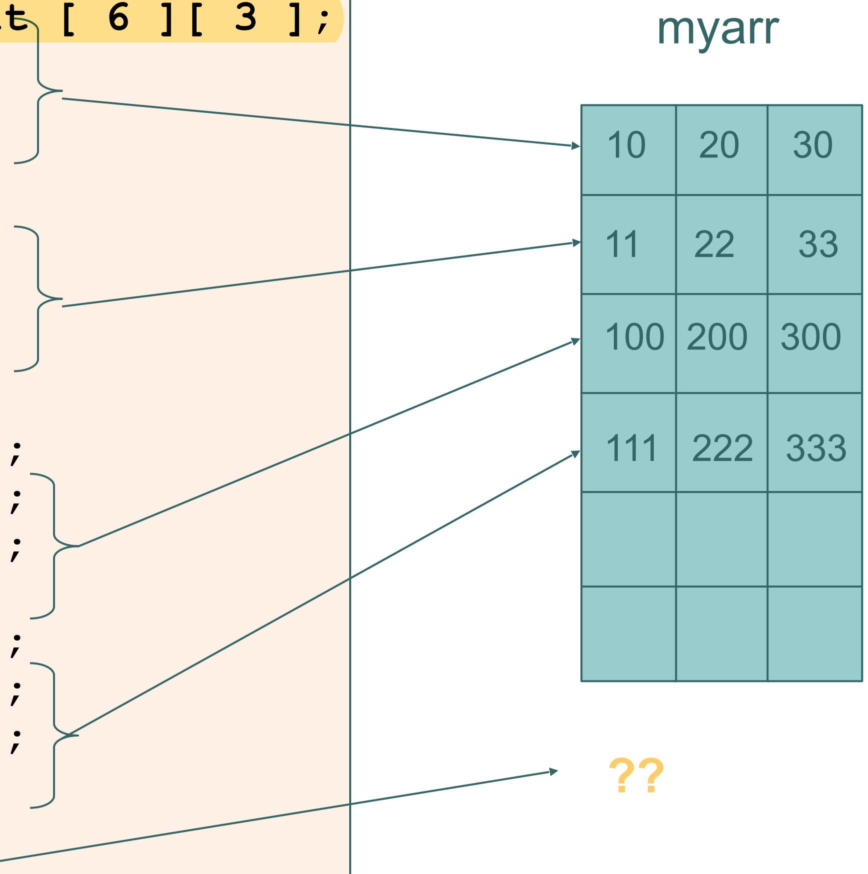
```
int myarr [ ] [ ]= new int [ 6 ][ 3 ];
myarray [ 0 ] [ 0 ] = 10;
myarray [ 0 ] [ 1 ] = 20;
myarray [ 0 ] [ 2 ] = 30;

myarray [ 1 ] [ 0 ] = 11;
myarray [ 1 ] [ 1 ] = 22;
myarray [ 1 ] [ 2 ] = 33;

myarray [ 2 ] [ 0 ] = 100;
myarray [ 2 ] [ 1 ] = 200;
myarray [ 2 ] [ 2 ] = 300;

myarray [ 3 ] [ 0 ] = 111;
myarray [ 3 ] [ 1 ] = 222;
myarray [ 3 ] [ 2 ] = 333;

myarray [ 5 ] [ 3 ] = 1;
```



Example

- Write a program that reverses arrays
- Write a program that lists areas in cities using a two dimensional array

Sabancı Üniversitesi

Primitive Type Casting

- When different types of variables are in operation we might need to cast the types to the result variable's type.
- Example:

long = int + int $\longrightarrow$ implicit cast occurs from int to long

Following won't compile:

byte = int + int $\longrightarrow$ explicit cast from int to byte required

int = double + double $\longrightarrow$ explicit cast from double to int required

```
double d1 =2344.2333;
double d2 = 12233.4333;
int sum = (int)(d1 + d2);
```

When larger types are casted to smaller types (ex: double→int) larger type might loose precision

Sabancı Üniversitesi

A Rule of Thumb

- A literal integer is always "int"
- The result of an expression involving anything int sized or smaller is always int

```
byte b1 = 12;
byte b2 = 13;
byte b3 = b1 + b2;

byte c = 21 + 12;
```

Compiler error! Explicit cast required

Compiles, imlicit cast

About the Sign Bit

- First bit in binary representation of all types is “sign” bit (1:negative, 0:positive)
- All Java variables types are signed using the rule “Two’s complement”
- When explicit cast occurs from a bigger type to smaller type (ex int -> byte) the excessive bits from the left are removed

```
byte a = (byte)128;
System.out.println(a);

output:
1 ?
-128 ?
```

Sabancı Üniversitesi

Two’s Complement Representation

Sign Bit

0	1	1	1	1	1	1	1

- 8 Bits can represent 256 values
  - Unsigned base 10 values (0 thru 255)
  - Signed base 10 values (-128 thru 127) Zero is part of the positive set
  - “Sign Bit” uses “0” for positive, “1” for negative

0	0	0	0	0	0	0	0	Zero
0	0	0	0	0	0	0	1	+1
0	1	1	1	1	1	1	1	+127
1	0	0	0	0	0	1	0	-126
1	1	1	1	1	1	1	1	-1