Variables, Types, Operations on Numbers

CSE 1310 – Introduction to Computers and Programming
Vassilis Athitsos
University of Texas at Arlington

Declaring a Variable

- At any point, you can create a variable, by doing a **variable declaration**.
- There are two ways to declare a variable: you can simply declare the type and name, or you can also provide an initial value.

  type variable_name;

  type variable_name = initial_value;

- For example:

```
int x = 123;
int number_of_fingers = 5;
double radius = 20.231;
```

Declaration/Initialization before Use

- A variable must be declared before we try to use it.
- The code below illustrates a common mistake.
  - Variable var is not declared anywhere.
- Java will refuse to run this code, complaining that it "cannot find symbol var".

```
public class example1 // incorrect code
{
  public static void main(String[] args)
  {
    var = 5;
    System.out.println(var);
  }
}
```

Declaration/Initialization before Use

- A variable must be declared and initialized before we try to use it.
- The code below illustrates a common mistake.
  - Variable **var** is declared but not initialized.
- Java will refuse to run this code, complaining that "variable var might not have been initialized".

```
public class example1 // incorrect code
{
  public static void main(String[] args)
  {
    int var;
    System.out.println(var);
  }
}
```

Declaration/Initialization before Use

- One way to fix such problems is to provide an initial value for the variable at the same line where you declare the variable.
- The code below shows an example of doing that.
  - The line shown in red declares and initializes a variable called **var**.

```
public class example1 // correct code
{
  public static void main(String[] args)
  {
    int var = 5;
    System.out.println(var);
  }
}
```

Declaration/Initialization before Use

- Another way is to first declare the variable in one line, and then set the value of the variable in another line.
- The code below shows an example of doing that.
  - The first line shown in red declares a variable called **var**.
  - The second line shown in red sets the value of **var** to 5.

```
public class example1 // correct code
{
  public static void main(String[] args)
  {
    int var;
    var = 5;
    System.out.println(var);
  }
}
```

Using Variables

```
public class hello1   // incorrect code
{
  public static void main(String[] args)
  {
    radius = 20.231;
    area = Math.PI * Math.pow(radius, 2);
    System.out.println(area);
  }
}
```

- What is wrong with this code?

Using Variables

```
public class hello1   // incorrect code
{
  public static void main(String[] args)
  {
    radius = 20.231;
    area = Math.PI * Math.pow(radius, 2);
    System.out.println(area);
  }
}
```

- What is wrong with this code?
  - Variables "radius" and "area" are not declared.

Using Variables

```
public class hello1 // correct code
{
  public static void main(String[] args)
  {
    double radius = 20.231;
    double area = Math.PI * Math.pow(radius, 2);
    System.out.println(area);
  }
}
```

- Corrected version.

Using Variables

```
public class hello1 // incorrect code
{
  public static void main(String[] args)
  {
    double area = Math.PI * Math.pow(radius, 2);
    double radius = 20.231;
    System.out.println(area);
  }
}
```

- What is wrong with this code?

Using Variables

```
public class hello1 // incorrect code
{
  public static void main(String[] args)
  {
    double area = Math.PI * Math.pow(radius, 2);
    double radius = 20.231;
    System.out.println(area);
  }
}
```

- What is wrong with this code?
  - Variable "radius" is used before it has been declared.

Using Variables

```
public class hello1 // correct code
{
  public static void main(String[] args)
  {
    double radius = 20.231;
    double area = Math.PI * Math.pow(radius, 2);
    System.out.println(area);
  }
}
```

- Corrected version.

Using Variables

```
public class hello1 // incorrect code
{
  public static void main(String[] args)
  {
    double radius = 20.231;
    double area = Math.PI * Math.pow(Radius, 2);
    System.out.println(area);
  }
}
```

- What is wrong with this code?

Using Variables

```
public class hello1 // incorrect code
{
  public static void main(String[] args)
  {
    double radius = 20.231;
    double area = Math.PI * Math.pow(Radius, 2);
    System.out.println(area);
  }
}
```

- What is wrong with this code?
  - Variable "radius" is misspelled in the line where the area is computed.

Using Variables

```
public class hello1 // correct code
{
  public static void main(String[] args)
  {
    double radius = 20.231;
    double area = Math.PI * Math.pow(radius, 2);
    System.out.println(area);
  }
}
```

- Corrected version.

Using Variables

```
public class example1 // incorrect code
{
  public static void main(String[] args)
  {
    int x = 5;
    int x = 3 * 5;
    System.out.println(x);
  }
}
```

- What is wrong with this code?

Using Variables

```
public class example1 // incorrect code
{
  public static void main(String[] args)
  {
    int x = 5;
    int x = 3 * 5;
    System.out.println(x);
  }
}
```

- What is wrong with this code?
  - Variable x is being declared twice.

Using Variables

```
public class example1 // correct code
{
  public static void main(String[] args)
  {
    int x = 5;
    x = 3 * 5;
    System.out.println(x);
  }
}
```

- Corrected version.

Types

- To declare a variable, you must specify the **type** of that variable.

  type variable_name;

  type variable_name = initial_value;

- The type of a variable defines what are **legal values** for that variable.
  - Java will never allow you to set a variable to a value incompatible with the type of the variable.

The Five Basic Types

- In this course, our main goal is to learn how to write programs that do basic processing of data.
  - The only data we care about in this course are numbers and text.
- To work with data and text, we will use five basic types.
- int
- double
- boolean
- String
- char

The Five Basic Types

- int
  - legal values? integers, like 0, 57, -1896…
- double
  - legal values? real numbers, like 3.0, 5.2, -0.23…
- boolean
  - legal values? only two: **true** and **false**.
- String
  - legal values? text, like "hello", "a cat jumped on the table", …
  - NOTE: text for strings must be enclosed in **double quotes**.
- char
  - legal values? singe characters, like 'c', '3', 'A', '#', …
  - NOTE: text for chars must be enclosed in **single quotes**.

Types Are **NOT** Interchangeable

- A common mistake for beginners in programming is to not pay attention to types.
  - Only beginners make this mistake.
  - You will not make it past beginner stage as long as you make this mistake.
- The following four values are NOT interchangeable:

2

2.0

"2"

'2'

- Why?

Types Are **NOT** Interchangeable

- A common mistake for beginners in programming is to not pay attention to types.
  - Only beginners make this mistake.
  - You will not make it past beginner stage as long as you make this mistake.
- The following four values are NOT interchangeable:

2 this is an int

2.0 this is a double

"2" this is a string

'2' this is a character

- Why? **Because they are different types**.

Types Are **NOT** Interchangeable

- For example:
  - If you write "2.5" when you should be writing 2.5, your code will not work.
  - If you write 5 when you should be writing '5', your code will not work.
  - If you write 2 when you should be writing 2.0, your code will not work.
  - If you write "true" when you should be writing true, your code will not work.

Types Are **NOT** Interchangeable

- For example:

Incorrect	Correct
String a1 = 2.5;	String a1 = "2.5";
double a2 = "2.5";	double a2 = 2.5;
int num = '5';	int num = 5;
char c1 = 5;	char c1 = '5';
String str = '5';	String str = "5";
int my_int = 2.0;	int my_int = 2;
boolean v = "true";	boolean v = true;
String v = true;	String v = "true";

The ++ and -- Operators

```
public class example1 {
  public static void main(String[] args) {
    double x = 5.5;
    x++;
    System.out.println(x);
    int y = 4;
    y--;
    System.out.println(y);
  }
}
```

```
Output

6.5
3
```

- The ++ operator increments the value of a variable by 1.
- Syntax:
  variable_name++;
- The -- operator increments the value of a variable by 1.
- Syntax:
  variable_name--;

The ++ and -- Operators

```
public class example1 {
  public static void main(String[] args) {
    double x = 5.5;
    x++;
    System.out.println(x);
    int y = 4;
    y--;
    System.out.println(y);
  }
}
```

```
Output

6.5
3
```

- The following two lines do the EXACT SAME THING:

variable_name++;

variable_name = *variable_name* + 1;

- The following two lines do the EXACT SAME THING:

variable_name--;

variable_name = *variable_name* - 1;

The ++ and -- Operators

```
public class example1 {
  public static void main(String[] args) {
    double x = 5.5;
    x = x+1;
    System.out.println(x);
    int y = 4;
    y = y-1;
    System.out.println(y);
  }
}
```

```
Output

6.5
3
```

- An alternative version of the previous program, without using ++ and --.
- Whether you use ++ and -- or not is entirely up to you.
- However, you should understand what they do when you see them in code.

The += and -= operators

```
public class example1 {
  public static void main(String[] args) {
    double x = 5.5;
    x += 3.2;
    int y = 20;
    y -= 5;
    System.out.println(x);
    System.out.println(y);
  }
}
```

```
Output

8.7
15
```

- The += operator adds some value to a variable.
- Syntax:
  variable_name += *value*;
- The -= operator subtracts some value from a variable.
- Syntax:
  variable_name -= *value*;

The += and -= operators

```
public class example1 {
  public static void main(String[] args) {
    double x = 5.5;
    x += 3.2;
    int y = 20;
    y -= 5;
    System.out.println(x);
    System.out.println(y);
  }
}
```

```
Output

8.7
15
```

- The following two lines do the EXACT SAME THING:

variable_name += *value*;

variable_name = *variable_name* + *value*;

- The following two lines do the EXACT SAME THING:

variable_name -= *value*;

variable_name = *variable_name* - *value*;

The += and -= operators

```
public class example1 {
  public static void main(String[] args) {
    double x = 5.5;
    x = x + 3.2;
    int y = 20;
    y = y - 5;
    System.out.println(x);
    System.out.println(y);
  }
}
```

```
Output

8.7
15
```

- An alternative version of the previous program, without using += and -=.
- Whether you use += and -= or not is entirely up to you.
- However, you should understand what they do when you see them in code.

Multiple Ways to Add/Subtract 1

- If we want to add 1 to x, in how many ways can we do it?

- If we want to subtract 1 from x, in how many ways can we do it?

Multiple Ways to Add/Subtract 1

- If we want to add 1 to x, in how many ways can we do it?

```
x++;
x += 1;
x = x+1;
```

- If we want to subtract 1 from x, in how many ways can we do it?

```
x--;
x -= 1;
x = x-1;
```

Converting Doubles to Ints

```
public class example1 {
  public static void main(String[] args) {
    double price = 18.53;
    int dollars = price;

    System.out.printf("Rounded price: %d dollars", dollars);
  }
}
```

- The above code gives an error:

Converting Doubles to Ints

```
public class example1 {
  public static void main(String[] args) {
    double price = 18.53;
    int dollars = price;

    System.out.printf("Rounded price: %d dollars", dollars);
  }
}
```

- The above code gives an error:
  - Java does not allow assigning a double value to an int variable.
- There are several ways to get around that.

Converting Doubles to Ints

```
public class example1 {
  public static void main(String[] args) {
    double price = 18.53;
    int dollars = (int) price;

    System.out.printf("Rounded price: %d dollars", dollars);
  }
}
```

- First approach: casting.
  - Putting (int) in front of the double value asks Java to convert that value to an integer.
  - Casting simply removes the decimal part.
  - `(int) 18.53` evaluates to 18.
  - `(int) -18.53` evaluates to -18.

Converting Doubles to Ints

```
public class example1 {
  public static void main(String[] args) {
    double price = 18.53;
    int dollars = (int) Math.round(price);

    System.out.printf("Rounded price: %d dollars", dollars);
  }
}
```

- Second approach: rounding.
  - Math.round(*number*) rounds *number* to the closest integer.
  - We still need to put (int), to convert the result of Math.round into an integer.
  - `(int) Math.round(18.53)` evaluates to 19.
  - `(int) Math.round(-18.53)` evaluates to -19.

Converting Doubles to Ints

```
public class example1 {
  public static void main(String[] args) {
    double price = 18.53;
    int dollars = (int) Math.floor(price);

    System.out.printf("Rounded price: %d dollars", dollars);
  }
}
```

- Third approach: rounding down (taking the floor).
  - Math.floor(*number*) rounds *number* down to an integer.
  - We still need to put (int), to convert the result of Math.floor into an integer.
  - `(int) Math.floor(18.53)` evaluates to 18.
  - `(int) Math.floor(-18.53)` evaluates to -19.

Converting Doubles to Ints

```
public class example1 {
  public static void main(String[] args) {
    double price = 18.53;
    int dollars = (int) Math.ceil(price);

    System.out.printf("Rounded price: %d dollars", dollars);
  }
}
```

- Fourth approach: rounding up (taking the ceiling).
  - Math.ceil(*number*) rounds *number* up to an integer.
  - We still need to put (int), to convert the result of Math.ceil into an integer.
  - `(int) Math.ceil(18.53)` evaluates to 19.
  - `(int) Math.ceil(-18.53)` evaluates to -18.

Constant Variables

```
public class example1 {
  public static void main(String[] args) {
    int weeks = 12;
    final int days_per_week = 7;
    int days = weeks * days_per_week;
    System.out.printf("%d weeks = %d days\n", weeks, days);
  }
}
```

- Some variables should never change value.
- Examples:
  - Number of days in a week.
  - Mathematical constants such as pi, e.
  - Physics constants like Newton's constant for gravity.

Constant Variables

```
public class example1 {
  public static void main(String[] args) {
    int weeks = 12;
    final int days_per_week = 7;
    int days = weeks * days_per_week;
    System.out.printf("%d weeks = %d days\n", weeks, days);
  }
}
```

- If you want to tell Java that a variable is a constant, you use the **final** keyword when you declare the variable.
- Syntax:
  final *type variable_name* = *value*;

Constant Variables

```
public class example1 {
  public static void main(String[] args) {
    int weeks = 12;
    int days_per_week = 7;
    int days = weeks * days_per_week;
    System.out.printf("%d weeks = %d days\n", weeks, days);
  }
}
```

- What changes in the behavior of this program if we remove the **final** keyword?

Constant Variables

```
public class example1 {
  public static void main(String[] args) {
    int weeks = 12;
    int days_per_week = 7;
    int days = weeks * days_per_week;
    System.out.printf("%d weeks = %d days\n", weeks, days);
  }
}
```

- What changes in the behavior of this program if we remove the **final** keyword?
- Nothing.
- Then, why should we ever use it?

Constant Variables

```
public class example1 {
  public static void main(String[] args) {
    int weeks = 12;
    final int days_per_week = 7;
    int days = weeks * days_per_week;
    System.out.printf("%d weeks = %d days\n", weeks, days);
  }
}
```

- What changes in the behavior of this program if we remove the **final** keyword?
- Nothing.
- Then, why should we ever use it?
- It is good programming practice: it makes code easier to read and understand, and prevents human errors.

Example Where **final** Is Useful

```
public class example1 {
  public static void main(String[] args) {
    int days_per_week = 7;
    int weeks = 12;
    days_per_week++;
    int days = weeks * days_per_week;
    System.out.printf("%d weeks = %d days\n", weeks, days);
  }
}
```

- What do you think will happen with this code?

Example Where **final** Is Useful

```
public class example1 {
  public static void main(String[] args) {
    int days_per_week = 7;
    int weeks = 12;
    days_per_week++;
    int days = weeks * days_per_week;
    System.out.printf("%d weeks = %d days\n", weeks, days);
  }
}
```

- What do you think will happen with this code?
- Java will run it, and it will give the wrong answer (12 weeks have 96 days).
  - The days_per_week++ should not have happened, probably the programmer put it there by accident.
  - However, Java cannot possibly know that it was a mistake.

Example Where **final** Is Useful

```
public class example1 {
  public static void main(String[] args) {
    final int days_per_week = 7;
    int weeks = 12;
    days_per_week++;
    int days = weeks * days_per_week;
    System.out.printf("%d weeks = %d days\n", weeks, days);
  }
}
```

- What do you think will happen with this code?

Example Where **final** Is Useful

```
public class example1 {
  public static void main(String[] args) {
    final int days_per_week = 7;
    int weeks = 12;
    days_per_week++;
    int days = weeks * days_per_week;
    System.out.printf("%d weeks = %d days\n", weeks, days);
  }
}
```

- What do you think will happen with this code?
- Java will refuse to run it, will give an error:
  - A constant variable is not allowed to change.
- By declaring a variable as **final**, you tell Java that if you ever try to change it, you are probably making a mistake and it should not allow you.

Example Where **final** Is Useful

```
public class example1 {
  public static void main(String[] args) {
    final int days_per_week = 7;
    int weeks = 12;
    days_per_week++;
    int days = weeks * days_per_week;
    System.out.printf("%d weeks = %d days\n", weeks, days);
  }
}
```

- You will see more examples of this as you learn programming:
  - Programming languages give us some tools, so that we do not allow ourselves to make mistakes.
  - The ability to declare a variable as a constant is such a tool.
  - This way, at least some mistakes are easily caught and fixed.