



Xi'an Jiaotong-Liverpool University

西交利物浦大學

CPT111 Java Programming

Lecture 3

**Data Types, Math,
Client Input, Coding Style**

Welcome!

- Welcome to Lecture 3 !
- In this lecture we are going to learn
 - Data types
 - Math Object and Random Numbers
 - Program Input and Interactive Programs
 - Casting and converting between types
 - Good coding practice

Data Types

In the early days of computing, memory (RAM: Random Access Memory) was incredibly expensive. The first Apple Mac had 128kB. My phone has 4,194,304 times more memory than that.

In 1987 1GB (Gigabyte) of RAM cost USD \$50,000. In 1981, \$300,000.

So, early software programs and apps were very carefully engineered to use as little memory as possible. So data types, too, were designed to save memory; there was no point in saving a number 7 in a piece of memory big enough for 7 billion.

Java data types: integer data

type	size	range	used for
byte	8 bits	-128 to 127 (0 to 255)	Bytes. eg individual bytes of image data; in 32 bit images there is 1 byte each for red, green, blue and alpha (transparency)
short	16 bits	-32,768 to 32,767	To save memory – instead of int . However, in most JVMs they are 32-bit anyway, so useless.
int (default)	32 bits	-2^{31} to $2^{31} - 1$	Everything except very big/very negative values. This is the default.
long	64 bits	-2^{63} to $2^{63} - 1$	long bigNumber = 3000000000000000000L;

- Integers (to non-mathematicians) are whole numbers, numbers with no fractional part, no decimal point.
- You will probably use **int** 95%+ of the time, **long** sometimes, and **byte** occasionally.

2,147,483,647 (two billion, one hundred and forty-seven million, four hundred and eighty-three thousand, six hundred and forty-seven) is the highest possible value of an int.

How big? 1 integer is 32 bits: $2^{31} - 1$

If we start counting from 0 and add 1 every second, we will reach this number in 68 years, 19 days, 3 hours 14 minutes and 7 seconds.

What happens if we 'overflow'? Theoretically, it could be anything. In practice, the next second would go back to zero, and go on counting as if nothing had happened.

This can make overflow very difficult to detect and very dangerous.

So what if we want to store a bigger integer? We can use type **long**, which is 64 bits. Its maximum value is $2^{63} - 1$ or 9,223,372,036,854,775,807.

Java data types – floating point

type	size	range	accuracy
float	32 bits	$\pm 10^{38}$ approx.	7 significant digits (decimal) float f = 1.375f;
double	64 bits	$\pm 10^{308}$ approx.	15 significant digits (decimal) Default

- Floating point data types
 - These can be slightly inaccurate
 - Rounding errors can appear
 - 0.8 may become 0.7999999999

Java data types – numeric

type	size	range	used for
byte	8 bits	-128 to 127 (0 to 255)	Bytes. eg individual bytes of image data; in 32 bit images there is 1 byte each for red, green, blue and alpha (transparency)
short	16 bits	-32,768 to 32,767	To save memory – instead of int. However, in most JVMs they are 32-bit anyway, so useless.
int (default)	32 bits	-2^{31} to $2^{31} - 1$	Everything except very big/very negative values. This is the default.
long	64 bits	-2^{63} to $2^{63} - 1$	long bigNumber = 3000000000000000000L;
float	32 bits	$\pm 10^{38}$ approx.	7 significant digits (decimal) float f = 1.375f;
double	64 bits	$\pm 10^{308}$ approx.	15 significant digits (decimal) Default

- Try them out!

Java data types – numeric

type	size	range	used for
int (default)	32 bits	-2^{31} to $2^{31} - 1$	Everything except very big/very negative values. This is the default.
long	64 bits	-2^{63} to $2^{63} - 1$	long bigNumber = 3000000000000000000L;
float	32 bits	$\pm 10^{38}$ approx.	7 significant digits (decimal) float f = 1.375f;
double	64 bits	$\pm 10^{308}$ approx.	15 significant digits (decimal) Default

```
int a = 10000000000;  
long b = 10000000000000000000000001;  
double c = 31.32;  
float d = 31.01f;
```

Java data types - Strings

- As introduced last week, can also use Strings

- Strings are objects, and have many methods
 - Will be discussed in future weeks
- At basic level, can store text data
- Note, double quotes needed around string

```
String str = "This is a String";
```

- **char**: represents a character ('a', 'b', 'c', etc)

- A different data type
- includes non-Roman characters (Chinese, Cyrillic, Arabic, Greek etc)
- It is actually a 16 bit integer, range 0 - 65,535

Java data types - Boolean

- We can also store True/False information
- Boolean (means true or false)
- Discussed later in the module
- Cannot store anything except this
- Very useful for checking if conditions are met
- Its size varies; it only uses 1 bit, but can take up to 32 bits of memory!

```
boolean bool = false;
```

Assigning variables

- Assigning different types of variables
- We can declare and assign separately

```
int intNum;    // declare a variable (assigned a default value)
intNum = 22;   // assign a new value
```

- We can also do both together!

```
int intNum = 22;    // initialize
double doubNum = 22.0;
float floatNum = 12.3f;
String strData = "Twenty two";
boolean boolData = true;
char charVal = 'c';
```

Operators

Doing stuff to data

Arithmetic operators

Operator	Description
+	Addition operator (also used for joining Strings) (called String <i>concatenation</i>)
-	Subtraction operator
*	Multiplication operator
/	Division operator
%	Modulo (remainder) operator

Java Arithmetic

- Simplest case

- Can use integers
- Whole numbers only
- Can use numbers
- Save data in an integer variable

```
int result = 3 + 3;
```

- View output with a System.out

```
System.out.println(result);
```

Java Arithmetic

- Multiple variables

- Using hard coding of numbers (3+3) is not best practice
- Better to use variables

```
int num1 = 4;  
int num2 = 6;  
int result = num1 - num2;  
System.out.println(result);
```


Java Arithmetic

- Numeric Data Types

- Remember, integers can only use whole numbers
- Not good for working with fractional numbers

```
int num1 = 3;  
int num2 = 4;  
int result = num1 / num2 ;  
System.out.println(result);
```

result = 0

- Better to use floats or doubles

```
double num1 = 3;  
double num2 = 4;  
double result = num1 / num2 ;  
System.out.println(result);
```

result = 0.75

Java Arithmetic - Divs

- The Division “/” Operator with doubles

```
double num1 = 7;  
double num2 = 3;  
double result = num1 / num2;  
System.out.println(result);
```

```
result =  
2.3333333333333333  
5
```

- But with integers?

```
int num1 = 7;  
int num2 = 3;  
int result = num1 / num2;  
System.out.println(result);
```

```
result = 2
```

- Why? Because an integer can only store whole numbers
 - Anything after the decimal point is simply ignored
 - Can cause coding errors, but can also be useful...

Java Arithmetic - Modulo

- Using a div with integers will ignore the decimal point

```
int num1 = 7;  
int num2 = 3;  
int result = num1 / num2;  
System.out.println(result);
```

result = 2

- What if we want to calculate the remainder?
- **7/4 = 1 remainder 3**

```
int num1 = 7;  
int num2 = 4;  
int result = num1 % num2;  
System.out.println(result);
```

result = 3

- Can be useful if you want to know if something divides exactly

3 % 2 = 1, an odd number
4 % 2 = 0, an even number

Java Arithmetic - Brackets

- Using brackets means that it will be processed first
- Example below, no brackets, processes div first, then add

```
int num1 = 7;  
int num2 = 3;  
int result = num1 / num2 + num2;  
System.out.println(result);
```

result = 5

- Adding brackets means it will process (num2 + num2) first

```
int num1 = 7;  
int num2 = 3;  
int result = num1 / (num2 + num2);  
System.out.println(result);
```

result = 1

- Very important!

Operator Precedence

Operators	Precedence (order of execution)
postfix	<i>expr++ expr--</i>
brackets	()
multiplicative	* / %
additive	+ -

Always does high precedence before low precedence

So always does * / % before doing + or –

Brackets are always first

Arithmetic Examples

```
int result = 3 + 6 - 7;
```

```
double result = 3 * 8 ;
```

```
int result = 5 + 7 % 3;
```

```
double result = 3 / 8 - 7 * 2;
```

```
double result = 3 / (8 - 7) * 2;
```

```
double result = 3 + 8 - 4 / 2;
```

```
double result = 3 + (8 - 4) / 2;
```

Unary Operators

Shortcuts

Unary operators – have only 1 operand

Operator	Description
++	Increment operator; increments a value by 1
--	Decrement operator; decrements a value by 1

```
int a;  
a = 41;  
a++;  
System.out.println("a = " + a);  
a--;  
System.out.println("a = " + a);
```

Output:

```
a = 42  
a = 41
```

a++ is the same as **a = a + 1**
a-- is the same as **a = a - 1**

We also have:

a += 2;

a = a + 2;

a -= 3;

a *= 3;

a = a * 3;

Operator Precedence

Operators	Precedence (order of execution)
postfix	<i>expr++</i> <i>expr--</i>
multiplicative	* / %
additive	+ -

Math Object

Additional number operations

Maths

- What if we wish to do additional arithmetic?
- Square root? Random numbers?
- We can use the Math object
 - This is an object (will be covered later in the semester) that has a number of methods
 - These methods are blocks of code that can be used by you
- You can access them by typing:
`Math.`

Math - Examples

- Find the square root of a variable

```
int num1 = 5;  
double root = Math.sqrt(num1);  
System.out.println(root);
```

- Get the maximum of 2 numbers

```
int num1 = 5;  
double root = Math.max(num1, num2);  
System.out.println(root);
```

Random number

- `Math.random()` will generate a pseudo-random number
- Easy way to get a random number
 - will be between 0.0 and 1.0, but **not** including 1.0, in set symbol $[0.0, 1.0)$

```
double rand = Math.random();
```

- if you need a different range, is more complex
- Can also apply a multiplier
 - This example will produce a double between 0.0 and 99.9999

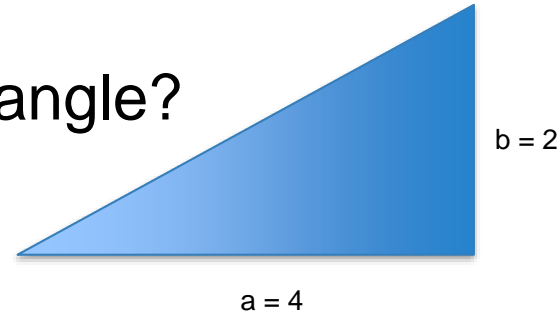
```
double rand = Math.random()*100;
```

Math – Other Examples

Method	Description	Return Type
acos(x)	Returns the arccosine of x, in radians	double
asin(x)	Returns the arcsine of x, in radians	double
atan(x)	Returns the arctangent of x between -PI/2 and PI/2 radians	double
exp(x)	Returns the value of E^x	double
expm1(x)	Returns $e^x - 1$	double
floor(x)	Returns the value of x rounded down to its nearest integer	double
hypot(x, y)	Returns $\sqrt{x^2 + y^2}$ without intermediate overflow or underflow	double
log(x)	Returns the natural logarithm (base E) of x	double
max(x, y)	Returns the number with the highest value	double float int long
min(x, y)	Returns the number with the lowest value	double float int long
pow(x, y)	Returns the value of x to the power of y	double
random()	Returns a random number between 0 and 1	double
round(x)	Returns the value of x rounded to its nearest integer	int
signum(x)	Returns the sign of x	double
sin(x)	Returns the sine of x (x is in radians)	double
sqrt(x)	Returns the square root of x	double
tan(x)	Returns the tangent of an angle	double
toDegrees(x)	Converts an angle measured in radians to an approx. equivalent angle measured in degrees	double
toRadians(x)	Converts an angle measured in degrees to an approx. angle measured in radians	double

Triangles - Hypotenuse

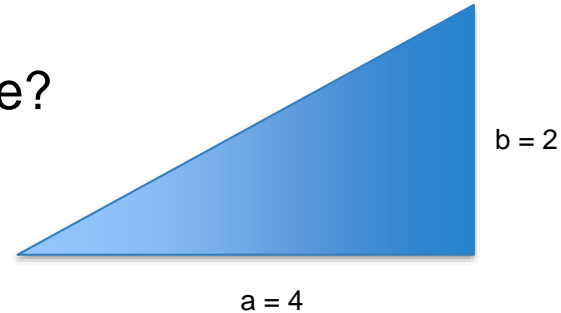
- Calculate the hypotenuse of a triangle?
- $a^2 + b^2 = c^2$
- $c = \sqrt{a^2 + b^2}$
- Can use `Math.pow` and `Math.sqrt`



Method	Description	Return Type
<code>hypot(x, y)</code>	Returns $\text{sqrt}(x^2 + y^2)$ without intermediate overflow or underflow	double
<code>pow(x, y)</code>	Returns the value of x to the power of y	double
<code>sqrt(x)</code>	Returns the square root of x	double

Triangles - Solution

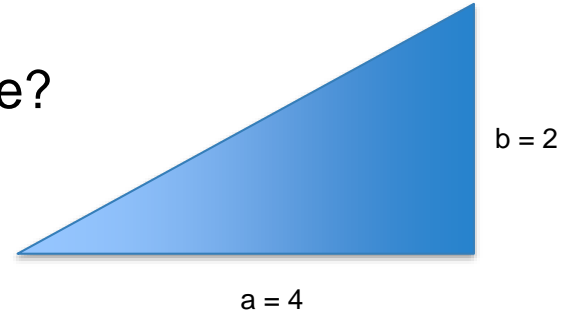
- Calculate the hypotenuse of a triangle?
- $c = \sqrt{a^2 + b^2}$
- $c = 4.47$



```
double a = 4;  
double b = 2;  
  
double c = Math.sqrt(Math.pow(a,2) + Math.pow(b,2));  
  
System.out.println("Value of C " + c);
```


Triangles – Simpler Solution

- Calculate the hypotenuse of a triangle?

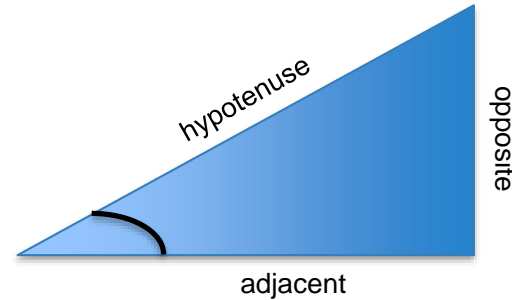


- Java also has the Math.hypot for exactly this purpose!

```
double a = 4;  
double b = 2;  
  
double simpleC = Math.hypot(a, b);  
System.out.println("Simpler Value of C " + simpleC);
```

Triangles - Angles

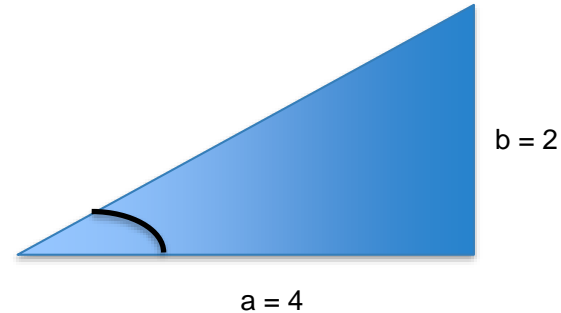
- Angles of a triangle?
- Can use SOHCAHTOA
- SOH = Sin – Opposite and Hypotenuse
- CAH = Cos – Adjacent and Hypotenuse
- TOA = Tan – Opposite and Adjacent



Method	Description	Return Type
<code>acos(x)</code>	Returns the arccosine of x, in radians	double
<code>asin(x)</code>	Returns the arcsine of x, in radians	double
<code>atan(x)</code>	Returns the arctangent of x between -PI/2 and PI/2 radians	double
<code>toDegrees(x)</code>	Converts an angle measured in radians to an approx. equivalent angle measured in degrees	double
<code>toRadians(x)</code>	Converts an angle measured in degrees to an approx. angle measured in radians	double

Triangles

- Angles of a triangle?
- We can use TOA:
 ○ Adjacent (4)
 ○ Opposite (2)
- Can use TOA
- $\tan^{-1}\left(\frac{2}{4}\right)$
- Here we can use the atan() method
- Will give us answer in radians, so need to convert to degrees



```
double opp = 2;  
double adj = 4;  
  
double val = Math.atan(opp/adj);  
double deg = Math.toDegrees(val);  
System.out.println("Angle is " + deg);
```

Introducing the Scanner

Input from Keyboard

Initial Programming

- Last week we made simple programs
- Everything in main method
- ints, doubles, Strings
- System.outs
- Not interactive

```
public static void main(String[] args) {  
    // A comment here  
    int num1 = 7;  
    int num2 = 3;  
    int result = num1 / num2 + num2;  
    double result2 = num1 / num2 + num2;  
    String name = "Andrew";  
  
    System.out.print("Result 1 is:" + result + " ");  
    System.out.println("Result 2 is:" + result2);  
    System.out.println("Name is " + name);  
}
```

Week1_lecture_test.Week1_lecture_test > main >

Week1_lecture_test (run) ⌘

run:
Result 1 is:5 Result 2 is:5.0
Name is Andrew
BUILD SUCCESSFUL (total time: 0 seconds)

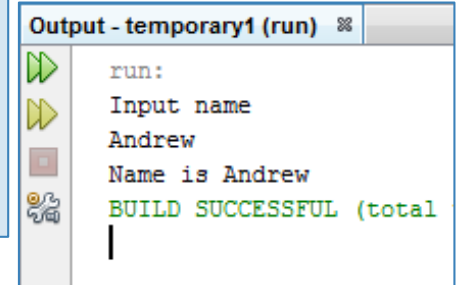
The Scanner Object

- The Scanner object is a useful way to receive the `System.input` character stream, i.e. accessing characters that you type.
 - Like a String, a Scanner is an object (we will discuss this in future weeks)
 - It contains a number of useful methods
- Its purpose is to receive and handle keyboard input
 - It is a very useful way to provide input to your program.
 - It wraps your character stream into a Scanner object, which has methods you can use to access it.

Scanner for keyboard

- We can create a new Scanner object easily:
Scanner kb = new Scanner(System.in);
- We can then use it to receive and store an input
String next = kb.nextLine();
- Useful to use a System.out to prompt user
- Input can be stored as a variable and used
System.out.println("Name is " + next);

```
Scanner kb = new Scanner(System.in);  
System.out.println("Input name");  
String next = kb.nextLine();  
System.out.println("Name is " + next);
```



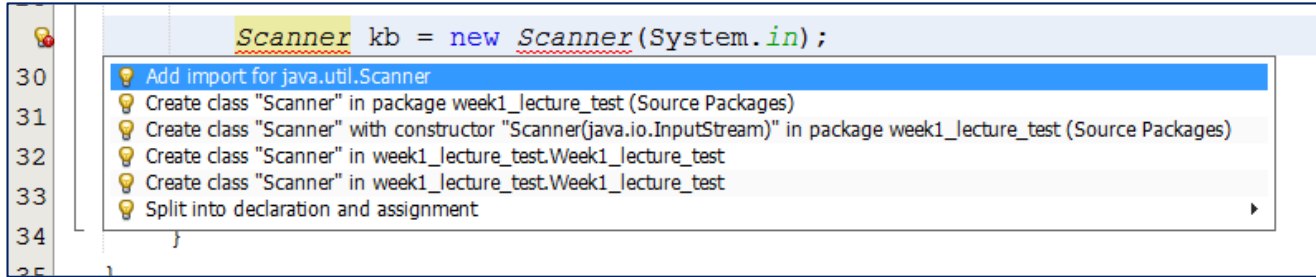
```
Output - temporary1 (run) %  
run:  
Input name  
Andrew  
Name is Andrew  
BUILD SUCCESSFUL (total  
|
```

Importing packages and classes

```
2 package hellostrings;
3
4 public class HelloStrings {
5     static Scanner keyboard;
6
7     public static void main(String[] args) {
8
9     }
```

- Scanner is an object, and we will discuss it in more depth in future weeks.
- For the next few weeks, all you need to know is that you can use it to get input from the keyboard.

Using Package



- You may notice an error message when you try to run it
- (red line under Scanner)
- This is because you need to tell Java that you want to use a Scanner object
- Easy to do, just click the little lightbulb and select the first option

Choose the first option ONLY!

```
import java.util.Scanner;


public class Week1_lecture_test {
    public static void main(String[] args) {
        // A comment here
        Scanner kb = new Scanner(System.in);
        System.out.println("Input name");
        String next = kb.nextLine();
        System.out.println("Name is " + next);
    }
}
```

```
package hellostrings;

import java.util.Scanner;

public class HelloStrings {
    static Scanner keyboard;

    public static void main(String[] args) {
        keyboard = new Scanner(System.in);
        System.out.println("Please input some text:");
        String input = keyboard.nextLine();
        System.out.println("You typed: "+input);
        keyboard.close();
    }
}
```

Output - HelloStrings (run) 



run:

Please input some text:

OK, here is some text

You typed: OK, here is some text

Input from Scanner

- Can be different types
- **next()**
 - Will receive a String (up to the first space)
 - `String next = kb.next();`
- **nextInt()**
 - Will receive an integer
 - `int next = kb.nextInt();`
- **nextDouble()**
 - Will receive a double
 - `double next = kb.nextDouble();`
- **nextLine()**
 - Will receive an entire line of text, including spaces
 - `String next = kb.nextLine();`

Please **DO NOT USE** `next()`, `nextInt()`,
`nextDouble()`.

We will **only use** `nextLine()`.

Interactive Programs

- Use a scanner to input a String
 - Store it as a String variable
 - Display the String
 - Our first interactive program!
-
- Can we use it for numbers too?

Interactive Programs 2

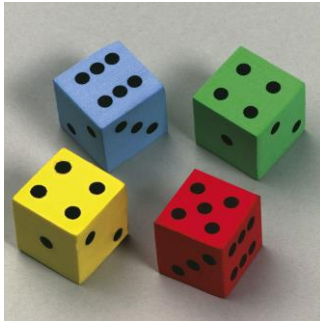
- More difficult to use for numbers `.nextLine()` returns a `String`, not a number
- One option, use different methods for different types
 - `kb.nextInt(); kb.nextDouble();` etc
 - Not recommended, problems with handling spaces, empty lines etc.
- Better option is to convert `Strings` to a number
 - How?
 - We can cast and convert!

Casting and Conversion

Changing a variable from one data type to another

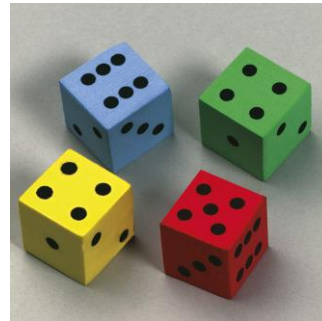
Some issues with numbers

- Rolling DICE We want to create a program that will roll some dice
- Will generate a random number between 1 and 6
- Our first (incorrect) attempt :



```
public static double getDice() {  
    double rand = Math.random() * 6+1;  
}
```


A problem?



- Data will be stored as a double value
 - 5.1, 4.2332
- We don't have dice that are 4.2332 sided!
 - We need to convert it to an integer.
- One way is to use some of the Math methods
 - Math.round(), Math.floor()
 - But this is still stored as a double, what if we want an integer?

```
public static double getDice() {  
  
    double floorRand = Math.floor((Math.random() * 6 ) + 1);  
  
}
```

We can cast our variables

- If we have a variable of one type
 - `double x = 1.53`
- And we want to convert it to a different type
 - i.e. an integer
- We can “cast” it
- Casting **converts** our variable from one type to another
- `int y = (int) x;`

Primitive Casting

- Casting is explicitly telling Java to make a conversion.
- Casts are required when you want to perform a narrowing conversion. You are instructing the compiler to convert.
- Narrowing runs the risk of losing information
 - i.e. losing accuracy
- The **cast** tells the compiler that you accept the risk.

Converting Primitives

Narrowing Conversion


Converting upwards, eg double to int, we will damage our data. This is called a narrowing conversion.



byte
char
short
int
long
float
double

Widening Conversion

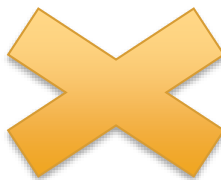
Converting downwards, eg int to double, we are not in danger of losing any data. This is called a widening conversion.



Primitive Casting

- We can already do some conversions:
- i.e. int to a double
- This is no problem, as we are “widening the conversion”
 - Making it more accurate
- But if we want to “narrow the conversion”
 - double to an integer
 - Causes compiler error!

```
int num = 3;  
double result =  
num;
```



```
double num =  
3.21;  
int result = num;
```

Casting

- In Java casting is the forcing of conversion between primitive data types, OR between objects of different classes.
- Use (int) to make an int data item from x. This is casting.
- We told the compiler to convert, information is lost.
- Casting does not “round” numbers

```
double x = 3.456;  
int y;  
y = (int) x;  
  
System.out.println(y);
```

Output → 3

```
double num = 3.412  
int res = (int) num;
```

Output → 3

```
double num = 3.999  
int res = (int) num;
```

Output → 3

Converting between primitive <-> String

- We need to do this a lot
 - For example, scanner input!
- There IS an Integer class and object.
- In fact, there are classes for ALL primitive data types
- They are called 'wrapper classes'.

Primitive values or Objects

Primitive data types

- boolean
- char
- byte
- short
- int
- long
- float
- double

Corresponding wrapper class

- Boolean
- Character
- Byte
- Short
- Integer
- Long
- Float
- Double

Using Wrapper Classes

- The main use for wrapper classes is to convert text to primitive number types

```
String s = "999"  
int res = Integer.parseInt(s);
```

```
String s = "99.021"  
double res = Double.parseDouble(s);
```

Using Wrapper Classes

- We can also convert from a primitive to a string!

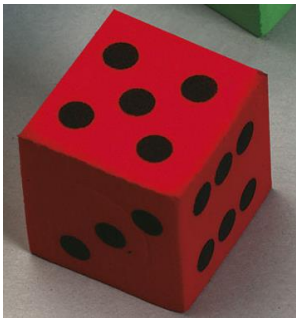
```
String s = "999"  
int res = Integer.parseInt(s);  
String out = Integer.toString(res);
```

```
String s = "99.021"  
double res = Double.parseDouble(s);  
String out = Double.toString(res);
```

```
boolean x = false;  
String m = Boolean.toString(x);
```

Casting and Conversions

- You can use casting to convert between types
- Not always recommended! Be careful
- There are usually Wrapper classes available that will let you convert as you need.



```
public static int getDice() {  
    double rand = (Math.random() * 6);  
    int r = (int) rand;  
    return r;  
}
```

- More generally (see Lab3), to generate a random integer between min and max:

```
randomInteger = min + (int)(Math.random() * (max - min + 1))
```

Scanner Input

- We had a problem with our scanner
- Using keyboard input, we could get input
- How to convert it to number?

```
Scanner kb = new Scanner(System.in);  
System.out.println("Input a number");  
String next = kb.nextLine();  
System.out.println("Number is " + next);
```

- This works for text, but not for numbers
- trying “int res = next+1;” causes errors

Scanner Input

```
public static void main(String[] args) {  
    Scanner kb = new Scanner(System.in);  
  
    System.out.println("Input a number");  
    String next = kb.nextLine();  
    int num1 = Integer.parseInt(next);  
}
```

- We can convert our String to an integer or double!
- Get input from Scanner
- Run a conversion, store as integer

Scanner Input

- A full program to take 2 numbers as input
- Will calculate results and display

```
public static void main(String[] args) {  
    Scanner kb = new Scanner(System.in);  
  
    System.out.println("Input a number");  
    String next = kb.nextLine();  
    int num1 = Integer.parseInt(next);  
  
    System.out.println("Input a number");  
    String next2 = kb.nextLine();  
    int num2 = Integer.parseInt(next2);  
  
    int res = num1+num2;  
    System.out.println("Total is " + res);  
}
```

Coding Style

Making beautiful code

Good programming style

- Your programming code must be easily read by others and by yourself.
- This saves time during program maintenance.
- It also leads to less human induced error.
- In software development projects, maintenance often accounts for 80% of effort.
- Program readability is of utmost importance.

Use meaningful names

```
String a1;  
int a2;  
double b;
```

Bad



```
String firstName;  
int dayOfWeek;  
double maxTemperature;
```

Good



Exceptions: some temporary variables such as loop counters.

Use meaningful names

Do not shorten words or remove vowels.

message

Good

msg

Bad

Capitalize first letter only in acronyms

importHtml

Good

importHTML

Bad

Use indentation

```
public static void main(String[] args) {  
  
    int square;  
    for (int i = 1; i <= 10; i++){  
        square = i * i;  
        System.out.print(i + " " + square);  
    }  
  
}
```

```
public static void main(String[] args) {  
  
    int square;  
    for (int i = 1; i <= 10; i++) {  
        square = i * i;  
        System.out.print(i + " " + square);  
    }  
  
}
```

Easier to understand.



**In NetBeans, Alt-Shift-F
Will auto indent for you.**

If using a text editor, indent
with spaces, NOT tabs.

Use whitespaces

```
public static void main(String[] args) {  
    int square;  
    for (int i=1;i<=10;i++) {  
        square=i*i;  
        System.out.print(i+" "+square);  
    }  
}
```

```
public static void main(String[] args) {  
  
    int square;  
    for (int i = 1; i <= 10; i++) {  
        square = i * i;  
        System.out.print(i + " " + square);  
    }  
  
}
```

Add blank lines to
Improve readability.

Put a space between each term.

```
public static void main(String[] args) {  
  
    int square;  
    for (int i = 1; i <= 10; i++) {  
        square = i * i;  
        System.out.print(i + " " + square);  
    }  
  
}
```

{ at the end of first line of block.

Placement of { and }

The code between { }
is called a block.

} on its own line at the end of a block.

Comments in your code

- Use `//` for a single comment.
- Use `/* */` for multiple lines.
- `//I have all my comments show in red`
- A one line comment to describe the purpose of a class or method might be enough.
- Add more comments if code is more complex.
- Well written code needs less comments.
- While you are learning, and on your own projects, comment everything!

Thank you for your attention !

- In this lecture, you have learned how to:
 - differentiate of integers, doubles, Strings
 - perform calculations in Java using the Math object
 - generate random numbers
 - compile and run a simple Java program
 - use a Scanner to read keyboard input
 - cast between different types of primitives
 - convert using Strings and wrapper classes
- Please continue to Lab 3, and solve
 - Exercise #3.1, #3.2, and
 - CW1 #3.1, #3.2