AP Computer Science A

```
UNIT 1 TOPIC 5
Casting & Ranges of Variables
```

College Board Alignment Unit 1 Topic 5



Primitive Types

1.5 Casting and Ranges of Variables

Determine the result or output based on statement execution order in a code segment without method calls (other than output).

5.B Explain why a code segment will not compile or work as intended.

ENDURING UNDERSTANDING

CON-1

The way variables and operators are sequenced and combined in an expression determines the computed result.

LEARNING OBJECTIVE

CON-1.C

Evaluate arithmetic expressions that use casting.

ESSENTIAL KNOWLEDGE

CON-1.C.1

The casting operators (int) and (double) can be used to create a temporary value converted to a different data type.

CON-1.C.2

Casting a double value to an int causes the digits to the right of the decimal point to be truncated.

CON-1.C.3

Some programming code causes int values to be automatically cast (widened) to double values.

CON-1.C.4

Values of type double can be rounded to the nearest integer by (int)(x + 0.5) or (int)(x - 0.5) for negative numbers.

CON-1.C.5

Integer values in Java are represented by values of type int, which are stored using a finite amount (4 bytes) of memory. Therefore, an int value must be in the range from Integer.MIN_VALUE to Integer.MAX_VALUE inclusive.

CON-1.C.6

If an expression would evaluate to an int value outside of the allowed range, an integer overflow occurs. This could result in an incorrect value within the allowed range.

32 bits and each bit is either a 1 or a 0 (2 values), so the largest possible number 2^32, or **4,294,967,296**

32 bits and each bit is either a 1 or a 0 (2 values), so the largest possible number 2^32, or **4,294,967,296**

Let's assume one of those 32 bits is "reserved" to indicate whether that number is positive or negative, so really we have 31 bits. What's the maximum value now?

32 bits and each bit is either a 1 or a 0 (2 values), so the largest possible number 2^32, or **4,294,967,296**

Let's assume one of those 32 bits is "reserved" to indicate whether that number is positive or negative, so really we have 31 bits. What's the maximum value now? 4,294,967,296 / 2 = 2,147,483,648

32 bits and each bit is either a 1 or a 0 (2 values), so the largest possible number 2^32, or **4,294,967,296**

Let's assume one of those 32 bits is "reserved" to indicate whether that number is positive or negative, so really we have 31 bits. What's the maximum value now? 4,294,967,296 / 2 = 2,147,483,648

Since we can hold negatives, what's the largest negative value?

32 bits and each bit is either a 1 or a 0 (2 values), so the largest possible number 2^32, or **4,294,967,296**

Let's assume one of those 32 bits is "reserved" to indicate whether that number is positive or negative, so really we have 31 bits. What's the maximum value now? 4,294,967,296 / 2 = 2,147,483,648

Since we can hold negatives, what's the largest negative value?

- 2,147,483,648

Agenda

- Demo: Casting & Ranges of int
- U1T5 Lab 1 (due next class)

Tomorrow:

- U1L5 Partner Programming Challenges
- Unit 1 Progress Check MCQ

DEMO: Casting & ranges of int

- In most programming languages, you may "cast" one data type into another.
- In Java, we have casting operators: (int) (double)
- A casting operator "converts" the item **directly to the right** of the notation

- In most programming languages, you may "cast" one data type into another.
- In Java, we have casting operators: (int) (double)
- A casting operator "converts" the item **directly to the right** of the notation

```
example 1 (int) 4.8
```

- In most programming languages, you may "cast" one data type into another.
- In Java, we have casting operators: (int) (double)
- A casting operator "converts" the item **directly to the right** of the notation



example]

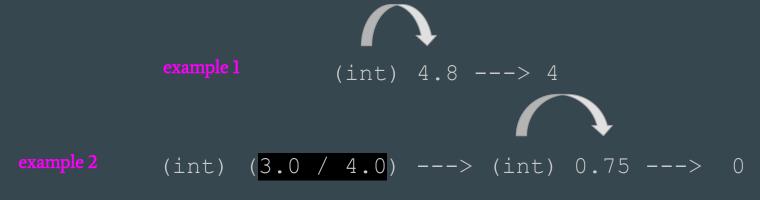
the (int) operator casts a double to an int value by *truncating* the decimal (it does **not** round); removing the decimal component results in a value that can be stored in 32 bits (the amount of memory set aside for int values)

- In most programming languages, you may "cast" one data type into another.
- In Java, we have casting operators: (int) (double)
- A casting operator "converts" the item directly to the right of the notation

- In most programming languages, you may "cast" one data type into another.
- In Java, we have casting operators: (int) (double)
- A casting operator "converts" the item **directly to the right** of the notation

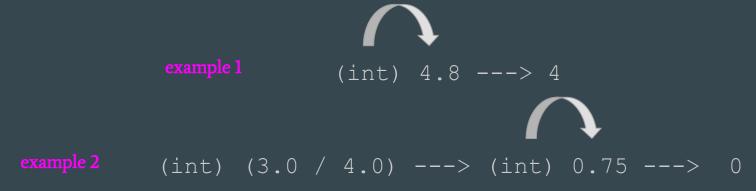


- In most programming languages, you may "cast" one data type into another.
- In Java, we have casting operators: (int) (double)
- A casting operator "converts" the item directly to the right of the notation

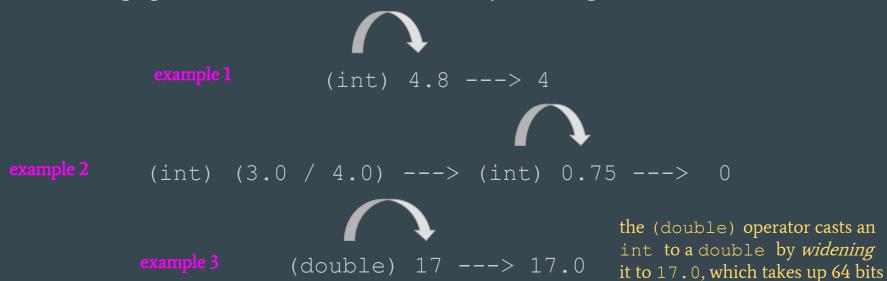


since (3.0 / 4.0) is in parentheses, it evaluates *first* to 0.75, and *then* the (int) operator casts 0.75 to the int value of 0 (truncation!)

- In most programming languages, you may "cast" one data type into another.
- In Java, we have casting operators: (int) (double)
- A casting operator "converts" the item **directly to the right** of the notation



- In most programming languages, you may "cast" one data type into another.
- In Java, we have casting operators: (int) (double)
- A casting operator "converts" the item **directly to the right** of the notation



```
example 5 (double) 3 / 4
```





example 5

(double) 3 / 4 ---> 3.0 / 4 ---> 0.75

the (double) operator casts the 3 (and only the 3) to a double first, and *then* the division takes place; the casting happens first because 3 / 4 is *not* in parentheses!

```
example 5 (double) 3 / 4 ---> 3.0 / 4 ---> 0.75

example 6 (double) (3 / 4)
```

```
example 5 (double) 3 / 4 ---> 3.0 / 4 ---> 0.75

example 6 (double) (3 / 4) ---> (double) 0 --->
```

```
example 5 (double) 3 / 4 ---> 3.0 / 4 ---> 0.75

example 6 (double) (3 / 4) ---> (double) 0 ---> 0.0
```

since (3 / 4) is in parentheses, it evaluates *first* to 0 (int/int division!), and *then* the (double) operator casts the int value 0 to the double value of 0.0

Order of operations when casting

First evaluate anything inside parentheses

Then perform casting operators

Then evaluate the expression!

Order of operations when casting

First evaluate anything inside parentheses

Then perform casting operators

Then evaluate the expression!

Try this one:

$$(double)(10 / 4) + 0.5 + (int) 6.7$$

Order of operations when casting

First evaluate anything inside parentheses

Then perform casting operators

Then evaluate the expression!

Try this one:

```
(double)(10 / 4) + 0.5 + (int) 6.7
```

```
(double) (10 / 4) + 0.5 + (int) 6.7
```

```
(double) (10 / 4) + 0.5 + (int) 6.7 // parentheses first
```

```
(double) (10 / 4) + 0.5 + (int) 6.7 // parentheses first --> (double) 2 + 0.5 + (int) 6.7
```

```
(double) (10 / 4) + 0.5 + (int) 6.7  // parentheses first
--> (double) 2 + 0.5 + (int) 6.7  // casting next
```

```
(double)(10 / 4) + 0.5 + (int) 6.7  // parentheses first
--> (double) 2 + 0.5 + (int) 6.7  // casting next
--> 2.0  + 0.5 + 6
```

```
(double)(10 / 4) + 0.5 + (int) 6.7  // parentheses first
--> (double) 2 + 0.5 + (int) 6.7  // casting next
--> 2.0  + 0.5 + 6  // evaluate
```

```
(double) (10 / 4) + 0.5 + (int) 6.7  // parentheses first

--> (double) 2 + 0.5 + (int) 6.7  // casting next

--> 2.0  + 0.5 + 6  // evaluate

--> 2.5 + 6

--> 8.5 (recall double + int gives a double!)
```

casting a double variable to an int

```
int someInt = 45;
double someDouble = 42.83;
someInt = someDouble;
```

casting a double variable to an int

casting a double variable to an int

```
int someInt = 45;
double someDouble = 42.83;
//someInt = someDouble;
someInt = (int) someDouble; // casting someDouble to an int
                                                   someInt = 42
System.out.println("someInt = " + someInt);
                                                  someDouble = 42.83
System.out.println("someDouble = " + someDouble);
```

casting an int variable to a double

```
int someInt = 45;
double someDouble = 42.83;

someDouble = (double) someInt; // casting someInt to a double

System.out.println("someInt = " + someInt);

System.out.println("someDouble = " + someDouble);

someInt = 42
someDouble = 42.0
```

casting an int variable to a double

```
int someInt = 45;
double someDouble = 42.83;

someDouble = (double) someInt; // casting someInt to a double
System.out.println("someInt = " + someInt);
System.out.println("someDouble = " + someDouble);
someInt = 42
someDouble = 42.0
```

However, casting an int to a double *isn't explicitly required*, since Java can hand that conversion automatically since doubles are 64 bits, bigger than an ints at 32 bits:

Range of int

- The maximum value that can be stored in a 32-bit int is $(2^32 1) = 2,147,483,647$ (technically not 2,147,483,648, long story why 1), and the minimum value is $-(2^32) = -2,147,483,648$
- Both of these are stored in Java as **constants**: **Integer.MAX_VALUE** and **Integer.MIN_VALUE**:

```
int maxInt = Integer.MAX_VALUE;
int minInt = Integer.MIN_VALUE;
System.out.println("max int = " + maxInt);
System.out.println("min int = " + minInt);
min int = -2147483648
```

• If you accidentally exceed the max or drop below the min in your program, you get a **runtime overflow error** -- not an exception (crash) -- which produces odd an unexpected results because values "wrap around"