Introduction to Type Conversion

JAVA Types in the AP Subset:

First let us review the 4 primitive data types covered in the AP JAVA subset:

- `int` – A whole number (or integer) ranging from -2,147,483,648 to 2,147,483,647.
- `double` – A floating point number with a VERY big range.
- `boolean` – Must be either `true` or `false`.
- `String` – A collection of alphanumeric characters.

> JAVA LITERALS:
> A literal is a value represented directly in your code. You have seen literals used frequently when assigning an initial value to a variable:
> `int x=5;`
> The number `5` is an `int` literal.
> `double y=5.0;`
> Since the literal value `5.0` has a decimal in it, it has the type `double`.
> A `boolean` variable must have either the literal value `true` or `false`.
> `String s="5.0";`
> Since the literal value `"5.0"` has quotes around it, it is a `String` literal.

When we assign a value to a variable, the value assigned must match the type of the variable. For example:

```
boolean b=true;
```

This statement is okay because the variable `b` has the type `boolean` and it is assigned the literal value `true`. But what happens if the types don't match:

```
boolean b=5;
```

The literal value `5` has the type `int` because it has no decimal, so it cannot legally be assigned to the variable `b` because it is a `boolean` variable!

In general the type of value assigned to a variable must match the type of the variable itself.

Widening Conversion:

JAVA, like most programming languages, allows variable types to be changed through a process called TYPE CONVERSION.

The first kind of type conversion we will examine is WIDENING PRIMITIVE CONVERSION. This is an automatic conversion from a more limited primitive data type to a "wider" primitive type. A wider primitive type is one that can store the information of another type without the loss of any information.

Thus, a `double` is a wider primitive data type than an `int`, so it is legal to write:

```
double d=5;
```

Because `5` is an `int` literal, but a `double` can hold the value represented by an `int` without losing data. The variable `d` has the value `5.0`!

However, trying to convert from a double to an int automatically is not allowed. For example:

```
int i=5.5;
```

This statement is **not** allowed because the `.5` portion of the `double` literal value `5.5` would be lost if the value were converted to an `int`!

Conversion in Expressions:

An automatic type conversion can also occur as part of an expression. For example:

```
double d=5 + 1.2;
```

In this statement we have two primitive literals. The first is `5`, an `int` literal and the second is `1.2` that is a `double` literal. In order to add these two literal values, the `int` literal `5` will first be converted to the wider `double` literal value `5.0`. Then the expression can be evaluated because all the types involved match. Finally, the resulting value `6.2` will be assigned to the variable `d`!

String Conversion in Expressions:

A `String` conversion will occur if the value on either side of the + operator is a `String` type:

```
String s=5 + "1.2";
```

Again, in this statement we have two primitive literals. The first is `5`, an `int` literal and the second is `"1.2"` that is a `String` literal. In order to add these two literal values, the `int` literal `5` will first be converted to a `String` literal value `"5"`. Then the expression can be evaluated because all the types involved match. The process of "adding" two String literals is called concatenation. In this example the resulting `String` literal value `"51.2"` will be assigned to the variable `s`.

`String` conversions can be confusing, so let's look at a few examples:

```
String a="One" + 2 + "Three";
```

Using substitution we can see that the expression `"One" + 2 + "Three"` will require that the `int` literal `2` be converted to a `String` so it reads:

```
"One" + "2" + "Three"
```

Now the three String literals can be combined into `"One2Three"`.

Multiple Conversion in Expressions:

Conversions occur left to right as the expression is evaluated.

Let's look at a complex expression including several conversions:

```
String b=1 + 1.0 + "Two" + true;
```

In this example, we first add the `int` literal `1` and the `double` literal `1.0`, so the `1` is widened through conversion to `1.0` then added to the double literal `1.0` resulting in the `double` literal value `2.0`. Next the `double` literal value `2.0` will be converted to a `String` because there is a `String` literal after the `+` operator so we have `"2.0"+"Two"` giving us the `String` literal `"2.0Two"`. Next the `boolean` literal `true` must be combined with the `String` literal `"2.0Two"` so the `boolean` literal is converted to the `String` literal value `"true"` and combined with the `String` literal `"2.0Two"` so we have the final `String` literal `"2.0Twotrue"` that is assigned to the `String` variable `b`.

Casting Conversion:

A variable can be forced to convert from one type to another through the process of CASTING.

To force a type to cast, the type you want it become is put in parenthesis before the expression:

```
int g=(int)5.5;
```

In this example, the `double` literal `5.5` is cast into an `int`. When a `double` is cast into an `int`, only the integer portion of the double is kept, so `g` is assigned the `int` literal `5`.

For now, the only two variable types that can be safely cast are a `double` into an `int` (like the example above) or an `int` into a `double` (though this is usually not necessary since expansion will occur automatically.

Example:

```
int h=(int)(10 * 5.25);
```

The parenthesis cause the expression to be evaluated first, so the `int` literal `10` is converted into the `double` literal `10.0` that is then multiplied by `5.25` resulting in a `double` value of `52.5`. Then the casting turns the `double` value `52.5` into the `int` value `52` which can be safely assigned to the `int` variable `h`.