

Introduction to Computer Science II

CS112-2012S-13

Type Casting and Midterm Review

David Galles

Department of Computer Science
University of San Francisco

13-0: Assignment and Types

- In java, an assignment statement

- `x = <expression>`

is legal if either:

- variable `x` and `<expression>` have the same type
 - Type of `<expression>` is a subclass of the type of `x`

13-1: Assignment and Types

```
int x;  
String y;  
Object z;
```

```
x = y;    // Legal?  
y = z;    // Legal?  
z = y;    // Legal?
```

13-2: Assignment and Types

```
int x;  
String y;  
Object z;
```

```
x = y;    // Not Legal!  
y = z;    // Not Legal!  
z = y;    // Legal!
```

13-3: Type Coercion

- It gets a little tricky with primitive types. Java will automatically convert (or coerce) an expression with lower precision to an expression with higher precision

```
double x = 0.0;  
int y = 1;
```

```
x = y; // Legal. Doubles can store any integer value  
y = x; // Not legal! Could lose information
```

13-4: Type Coercion

- It gets a little tricky with primitive types. Java will automatically convert (or coerce) an expression with lower precision to an expression with higher precision

```
double x = 0.0;  
float y = 2.3;
```

```
x = y; // Legal. Doubles can store any float value  
y = x; // Not legal! Could loose information
```

13-5: Type Coercion

- This can lead to some weirdness. Literal constants like 3.2 are assumed to be double-valued expressions, can assign them to float variables (since that could lose information)

```
float x = 0;    // OK, ints can be converted to floats  
float y = 0.0; // ERROR! 0.0 is a double value
```

- We can specify that we want a float constant using the 'f' prefix

13-6: Type Coercion

- This can lead to some weirdness. Literal constants like 3.2 are assumed to be double-valued expressions, can assign them to float variables (since that could lose information)

```
float x = 0;    // OK, ints can be converted to floats
float y = 0.0f; // OK, float valued constant
float z = 0.0;  // ERROR! 0.0 is a double value
```

13-7: Mixed Types

- If you use different types (int, float, double) in an operation that can handle both types (+, *), then:
 - Lower precision values are converted to higher precision values
 - Do the operation with the higher precision values

13-8: Mixed Types

- What are the values of the following expressions?
 - $3 / 2$
 - $3.0 / 2.0$;
 - $3 / 2.0$;
 - $3 / 2 * 4.0$

13-9: Mixed Types

- What are the values of the following expressions?
 - $3 / 2 == 1$ (int)
 - $3.0 / 2.0; == 1.5$ (double)
 - $3 / 2.0; == 1.5$ (double)
 - $3 / 2 * 4.0 == 4.0$ (double) Why?

13-10: Explicit Conversion

- Sometimes the automatic conversion might not work exactly as we want it to
- Sometimes we want to convert a higher precision value to a lower precision value (and we're OK with losing information)
- We can convert directly using a cast

13-11: Explicit Conversion

```
double z = 3.45;  
int x = (int) 3.0;  
float y = (float) z;
```

```
// Can still be tricky!  
double a = (double) 3 / 4;  
double b = (double) (3 / 4);
```

13-12: Explicit Conversion

```
double z = 3.45;  
int x = (int) 3.0;  
float y = (float) z;
```

// Can still be tricky!

```
double a = (double) 3 / 4;    // after assignment, a = 0.75  
double b = (double) (3 / 4); // after assignment, a = 0.0
```

13-13: Casting Classes

```
Object genericArray[] = new Object[3];  
genericArray[0] = "hello";  
genericArray[1] = "there";  
genericArray[2] = "MoreStrings!";
```

- What if we wanted to actually do something with one of these strings?
 - Get size, character at a particular index, etc

13-14: Casting Classes

```
Object genericArray[] = new Object[3];  
genericArray[0] = "hello";  
genericArray[1] = "there";  
genericArray[2] = "MoreStrings!";
```

```
String s = genericArray[0]; // Not Valid -- can't assign superclass  
                             // value to subclass variable  
int x = genericArray[0].length() // Not Valid -- while the data stored  
                                 // in genericArray[0] is a string (with  
                                 // a size method), Objects dont' have length
```

13-15: Casting Classes

```
Object genericArray[] = new Object[3];  
genericArray[0] = "hello";  
genericArray[1] = "there";  
genericArray[2] = "MoreStrings!";
```

```
String s = (String) genericArray[0];           // OK  
int x = s.length();                           // OK  
int y = ((String) genericArray[0]).length()    // OK
```

13-16: Casting Classes

- Array of objects
 - Store **any** object in the array
 - Can't *do* anything with the elements in the array without casting
 - (Can do some things – toString, for instance)

13-17: Casting Classes

- Following compiles just fine ...

```
Object o1 = "StringValue";  
Object o2 = new Integer(5);
```

```
String s1 = (String) o1;  
String s2 = (String) o2;
```

13-18: Casting Classes

- Following compiles just fine ...

```
Object o1 = "StringValue";
```

```
Object o2 = new Integer(5);
```

```
String s1 = (String) o1;
```

```
String s2 = (String) o2; // ClassCastException here : o2 is not a string
```

13-19: Midterm Review

- Arrays
- Inheritance
- Casting
- Method Overloading (operators and user defined)

13-20: Review: Arrays

- Arrays
 - All elements must be of the same type (but type can be Object!)
 - Can't change the size of an array (how do we work around this?)
 - Need to call “new” on array to create space

13-21: Review: Arrays

- Writing a tile-based video game (think original warcraft / starcraft / civilization / etc)
- Map is a 40 x 40 grid of tiles (forest tile, water tile, grassland tile, etc)
- Tile constructor takes as input parameter the type of tile – “forest”, “water” etc
 - `Tile t = new Tile("forest");`
- Give a block of code that creates a map of all forest tiles

13-22: Review: Arrays

- Give a block of code that creates a map of all forest tiles

```
Tile map[] = new Tile[40][40];  
for (int i = 0; i < map.length; i++)  
    for (int j = 0; j < map[i].length; j++)  
        map[i][j] = new Tile("forest");
```

13-23: Review: Arrays

- Give a block of code that creates a map with all interior forest, with a 1-unit wide strip of mountains surrounding the forest (examples on whiteboard)

13-24: Review: Arrays

- Give a block of code that creates a map with all interior forest, with a 1-unit wide strip of mountains surrounding the forest (examples on whiteboard)

```
Tile map[] = new Tile[40][40];
for (int i = 1; i < map.length; i++)
    for (int j = 1; j < map[i].length; j++)
        map[i][j] = new Tile("forest");
for (int i = 0; i < 40; i++)
{
    map[0][i] = new Tile("mountain");
    map[map.length - 1][i] = new Tile("mountain");
}
for (int i = 1; i < map.length; i++)
{
    map[i][0] = new Tile("mountain");
    map[i][map[i].length - 1] = new Tile("mountain");
}
```

13-25: Review: Arrays

- Give a block of code that creates a map with all interior forest, with a 1-unit wide strip of mountains surrounding the forest (examples on whiteboard)

```
Tile map[] = new Tile[40][40];
for (int i = 1; i < map.length; i++)
    for (int j = 1; j < map[i].length; j++)
    {
        if (i == 0 || i == map.length - 1 || j == 0 || j == map[i].length - 1)
        {
            map[i][j] = new Tile("mountain");
        }
        else
        {
            map[i][j] = new Tile("forest");
        }
    }
}
```

13-26: Review: Arrays

- Assume that Tile class has a method “type” that returns a string that describes the type of tile, “forest”, “mountain”, “water”, etc
- Give a method forrestSize that takes as input a map (2D array of Tiles) and returns the number of forest tiles in the map

13-27: Review: Arrays

```
int forestSize(Tile map[])
{
    int forestTiles = 0;
    for (int i = 0; i < map.length; i++)
    {
        for (int j = 0; j < map[i].length; j++)
        {
            if (map[i][j].equals("forest"))
            {
                forestTiles++;
            }
        }
    }
    return forestTiles;
}
```

13-28: Review: Arrays

```
public static int sum(int[] values)
{
    int sum = 0;
    for(int i = 0; i < values.length; i++)
    {
        sum += values[i];
        values[i] = 0;
    }
    return sum;
}

public static void main(String[] args)
{
    int[] tester = {1, 2, 3, 4, 5};
    System.out.println("Sum: " + sum(tester));
    for(int i = 0; i < tester.length; i++)
    {
        System.out.println "[" + i + "]: " + tester[i]);
    }
}
```

13-29: **Review: Inheritance**

- (Library example from Prof. Rollins)