CENG 211 – Programming Fundamentals

Methods, Objects, and Classes

Assist. Prof. Mustafa Özuysal
Contact: mustafaozuysal@iyte.edu.tr

2015-16 Fall Term

Classes and Objects

- When we define a new class we create a new type, which is a template for creating new variables called objects.
- A **constructor** is a special method with the class name that runs every time a new object is created to initialize it.
- You can create **many** objects of a **single** class.

```
import java.awt.Color;
public class Car {
  private Color color;
  public Car() { color = Color.PURPLE; }
  public void paint(Color new_color) {
    color = new_color;
  }
}
```

Car myCar = new Car();

Car yourCar = new Car();

color: Color.PURPLE

color: Color.PURPLE

Classes and Objects

- Each object will contain **all the variables and methods** that is part of the class definition (except static ones, see next slide).
- Each object will contain its **own copy** of the variables.

```
import java.awt.Color;
public class Car {
  private Color color;
  public Car() { color = Color.PURPLE; }
  public void paint(Color new_color) {
    color = new_color;
  }
}
```

yourCar.paint(Color.BROWN);

myCar
color: Color.PURPLE

yourCar
color: Color.BROWN

Classes and Objects

- **Methods** that are declared **static** are not called on the objects. They are called on the class.
- **Variables** that are declared **static** are not copied in the objects, there is only a single copy on the class.
- Static methods can only call static methods and access only static variables.

```
import java.awt.Color;
public class Car {
  private Color color;
  public Car() { color = Color.PURPLE; }
  public void paint(Color new_color) {
    color = new_color;
  }
}
  public static Color defaultColor() {
    return Color.PURPLE;
  }
}
```

yourCar.paint(Car.defaultColor());

myCar
color: Color.PURPLE

yourCar
color: Color.PURPLE

Classes and Objects

- Car.java

```
public class Car {
    private String color;

    public Car() {
        color = Car.defaultColor();
    }

    public void paint(String new_color) {
        color = new_color;
    }

    public void printColor() {
        System.out.print(color);
    }

    public static String defaultColor() {
        return "PURPLE";
    }
}
```

Classes and Objects

- CarTest.java

```
public class CarTest {
    private static void printCarColor(String carName, Car car) {
        System.out.print(carName + ": ");
        car.printColor();
        System.out.println();
    }

    public static void main(String[] args) {
            ……
    }
}
```

Classes and Objects

- CarTest.java

```
public class CarTest {
   private static void printCarColor(String carName, Car car) { …… }

   public static void main(String[] args) {
        Car myCar = new Car();
        Car yourCar = new Car();

        System.out.println("Initial colors:");
        printCarColor("myCar", myCar);
        printCarColor("yourCar", yourCar);

        System.out.println("Paint your car BROWN:");
        yourCar.paint("BROWN");
        printCarColor("myCar", myCar);
        printCarColor("yourCar", yourCar);

        System.out.println("Paint your car the default Car color:");
        yourCar.paint(Car.defaultColor());
        printCarColor("myCar", myCar);
        printCarColor("yourCar", yourCar);
   }
}
```

Classes and Objects

```
> javac Car.java
> javac CarTest.java
> java CarTest
Initial colors:
myCar: PURPLE
yourCar: PURPLE
Paint your car BROWN:
myCar: PURPLE
yourCar: BROWN
Paint your car the default Car color:
myCar: PURPLE
yourCar: PURPLE
```

Primitive Data Types

- **boolean**: Truth values, **true** and **false** are keywords that can be used to initialize these. Default value: false
- **byte**: 8 bit signed integers in the range [-128,127], default value: (byte)0
- **short**: 16 bit signed integers in the range [-32768, 32767], default value (short)0
- **int**: 32 bit signed integers in the range [-2147483648, 2147483647], default value 0
- **long**: 64 bit signed integers in the range [-9223372036854775808, 9223372036854775807], default value 0L
- **float**: Single-precision 32-bit floating point numbers. Default value 0.0f
- **double**: Double-precision 64-bit floating point numbers. Default value 0.0d
- **char**: 16 bit Unicode single characters like 'a' and '0'. Its value is between '\u0000' and '\uffff'. Default value '\u0000'
- There is no **unsigned** keyword in Java.

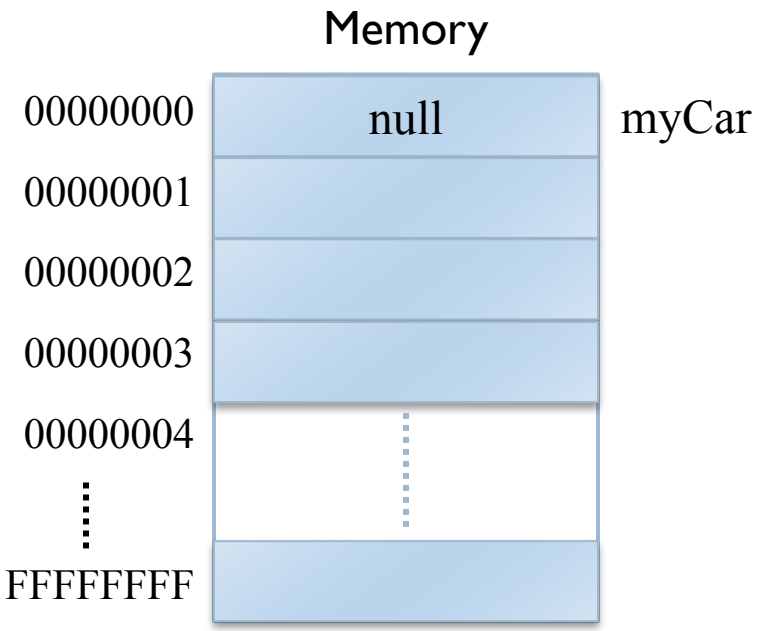
Reference Data Types

- All classes and arrays in Java are reference data types.
- There is a special reference value null that does not point to anything. All reference variables is initialized to null.
- Since String is a class in Java, String variables are also of reference type.

Reference Data Types

- When you first declare a variable of a reference type, compiler allocates memory to hold **only** the reference value and initializes it with null.

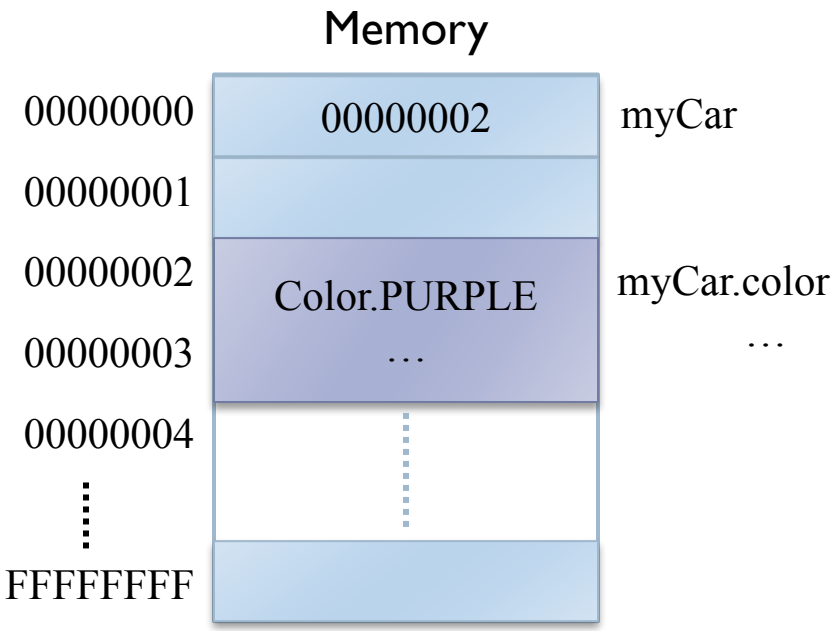
```
Car myCar;
```



Reference Data Types

- For the actual object, you need to allocate memory yourself with the new expression. Unlike in C, you do not need to free the memory afterwards, Java will free it if no reference exists to the allocated memory.

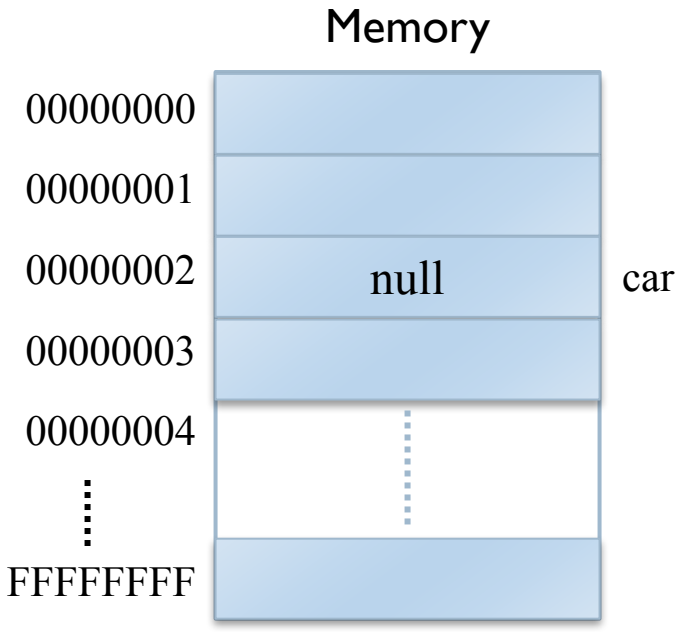
```
myCar = new Car();
```



Trying to reference null variables

```
public class NullCarTest {
    private static void printCarColor(String carName, Car car) {
            System.out.print(carName + ": ");
            car.printColor();
            System.out.println();
    }

    public static void main(String[] args) {
            printCarColor("myCar", null);
    }
}
```



car.printColor()????

```
> javac NullCarTest.java
> java NullCarTest
myCar: Exception in thread "main" java.lang.NullPointerException
        at NullCarTest.printCarColor(NullCarTest.java:4)
        at NullCarTest.main(NullCarTest.java:9)
```

Classes that box primitive variables

- For every primitive Java type there is a corresponding class that can store the same range of values:
  - Boolean
  - Byte
  - Short
  - Integer
  - Long
  - Float
  - Double
  - Character

Classes that box primitive variables

- You can create instances of these types with the new expression:

```
Integer bonus = new Integer(300);
```

- Or, you can directly assign primitive values:

```
Integer bonus = 300;
```

- These wrapper classes also have static variables and methods that provide extra functionality:

```
int maxInteger = Integer.MAX_VALUE;
int age = Integer.parseInt(ageString);
Integer age = Integer.valueOf(ageString);
int setBits = Integer.bitCount(flags);
```

String class

- All strings in Java are objects of class java.lang.String
- String objects have a handful of methods:
  - String s = “abcd”;
  - s.length(); // returns 4
  - s.startsWith(“ab”); // returns true
- String object contents are immutable
  - You can reassign a String variable but you can not change its contents in place:
  - String s = “a string”; // OK
  - s = “another string”; // OK
  - s[0] = “b”; // ERROR

String class

- String object contents are immutable
- As a result, string methods that needs to modify the string return a new string:
  - String s = “abcd”;
  - String sUpper = s.toUpperCase(); // sUpper ← “ABCD”
  - String s2 = s.substring(2); // s2 ← “cd”
  - String s1_2 = s.substring(1, 2); // s1_2 ← “b”
  - String se = s.replace(“c”, “e”); // se ← “abed”
- If you need mutable strings use the java.lang.StringBuffer class.
  - StringBuffer s = new StringBuffer(“abcd”);
  - s.append(“e”); // s ← “abcde”

Enhanced for statement

- To iterate over all elements of an array you can use the enhanced form of the for statement:

  for (<element-type> <loop-var>: <array>)

- This form also works with objects of certain classes. We will generalize it later.
- If the array reference is null, you will get an exception.
- Example:

```
int [] codes = { 211, 389, 461 };
for (int code: codes)
    System.out.println(String.valueOf(code));
```

```
> java ArrayTest
211
389
461
```

Command Line Arguments

- The array of strings that is passed as a parameter to the main function is filled in by the command line arguments:

```
public class ArgTest {
   public static void main(String[] args) {
      for (String arg: args)
         System.out.println(arg);
   }
}
```

```
> java ArgTest -g -Wall arg_test.c -o arg-test "has  spaces"
-g
-Wall
arg_test.c
-o
arg-test
has  spaces
```

Methods and Parameters

- You can declare methods inside classes similar to the way you define functions in C:

  <modifiers> <return-type> <method-name>(<parameters>) {

      <method-body>

  }

- Modifiers can be keywords like **static**, **public**, …
- We will see more modifiers as we progress.
- If the return type is void you can still return from a method by leaving the return expression empty:
  - return;

Methods and Parameters

- A method can take a variable number of arguments of a single type with the ellipsis (…) notation. The arguments are put into an array of the same type:

```
public static void main(String[] args) {
    System.out.println(String.valueOf(sum()));
    System.out.println(String.valueOf(sum(5)));
    System.out.println(String.valueOf(sum(5, 7)));
    System.out.println(String.valueOf(sum(5, 7, 10)));
}

public static int sum(int... numbers) {
    int s = 0;
    for (int num: numbers) {
        s += num;
    }
    return s;
}
```

```
> java ArgTest
0
5
12
22
```

Variable Scope

- The scope of a declaration is the portion of the code that the declared variable/type is visible in its simple form.
- Your variables are only visible in the block of code they are declared in.
- Method parameters are visible only inside the method.
- Class member variables are visible only within the class definition if they are private.
- A declaration in an inner scope can hide a variable with the same name in the outer scope.
- We will talk more about scopes while discussing encapsulation.