ALGONQUIN COLLEGE

METHODS: A DEEPER LOOK

Using SecureRandom class
Arguments Passing

Cyr M. Bakindé

Topics

- Random Class example
- Arguments passing
- Argument promotion and casting
- Method overloading
- Summary

Random Number Generation Example

```
import java.security.SecureRandom;
public class RandomTest {
        public static void main(String[] args) {
        SecureRandom ranNum = new SecureRandom();
        // rolling a six-sided dice six times
        int face = 0;
        for (int i =0;i<6;i++){
            face = 1 + ranNum.nextInt(6);
            System.out.println(face);
             }
       }
}
```

Arguments Passing

- Parameters – variables in the definition of methods or constructors
- Arguments – actual value of the variable that is passed to functions or constructors when calling them
- Two ways of passing arguments - by value or by reference
- By value – a copy of the argument's value (primitive or reference to an object) is passed to the called method. Change to the method's copy of the variable does not affect the original value
- By reference - the called method can access the argument's value directly and modify that data
- In Java, ***all arguments are passed by value***

Arguments Passing

- When object‘s reference is passed by value, the parameter in the called method and the argument in the calling method are the same
- Reference in the parameter and reference in the argument refer to the same object
- Objects themselves are not and cannot be passed

Ex:

```
class SampleClass{
    private int field = 5;
    public void setField(int newFieldval){
      this.field = newFieldval;
     }
    public String toString (){
        return ("field in object  " + field);
      }
}
```

Arguments Passing

```
public class ArgPassingTest {
    public static void main(String[] args) {
    //passing primitive variable
    int argVar= 5;
    System.out.println("argVar before method call " +argVar);
    printPrim(argVar);
    System.out.println("argVar after method call " +argVar);

    //passing reference variable
    SampleClass objVar = new SampleClass();
    SampleClass objVar1 = objVar;
    System.out.println(objVar);
    System.out.println(objVar1);
    printRef(objVar);
    System.out.println(objVar1);
   }
   public static void printPrim(int parVar){
   parVar += 2;
   System.out.println("primitive parVar is " +parVar);
   }
   public static void printRef(SampleClass parRef){
   parRef.setField(8);
   System.out.println(parRef);
   }
}
```

Argument Promotion and Casting

- Two types of data conversion in Java, widening conversion (promotion) and narrowing conversion
- Argument promotion is the widening conversion of the argument's data type to the data the method expects to receive
- Argument casting is the narrowing or widening conversion of the argument's data type to the data the method expects to receive
- Casting means explicitly telling Java to perform a conversion
- Casting can have surprising results
- Casts are required when performing narrowing conversion
- Four general cases of conversion and casting
  - Conversion of primitives
  - Casting of primitives
  - Conversion of object references
  - Casting of objects references

Argument Promotion and Casting

Type	Promotion Path
char	int, long, float and double
byte	short, int, long, float and double
short	int, long, float and double
int	long, float and double
long	float and double
float	double

Argument Promotion and Casting

```
public class Test {
   public static void main (String[] args) {
     char a = 5;
     byte b = 5;
     long c = 100;
     short d=200;

     //System.out.println(total);
     //Widening
      convert(d);
     //Casting with widening
      convert((int)b);
     //Casting with narrowing
     convert((int)c);
   }
    static void convert(int arg){
      System.out.println("Value passed is: " + arg   );
   }

}
```

Method Overloading

- The act of naming methods with different parameter using the same name in a class is call ***method overloading***
- Methods have ***signatures*** consisting of a combination the method's name, the number, types, and order of its parameters
- The compiler uses method signatures to distinguish between
- Method calls cannot be distinguished only by return type
- Ex:

```
void myMethod1(int a, float b){}
void myMethod1(float a, int b){}

void myMethod2(int a, float b){}
int myMethod2(int a, float b){} --  Error indicating method is already defined
```

set and *get* methods

- ***set*** and ***get*** of methods are used to manipulate instance data or fields. ***set*** methods are often called mutators because they change the values of fields. ***get*** methods are called accessor because they retrieve the values of fields. ***set*** and ***get*** methods should adhere to the following principles:
- The fields that ***set*** and ***get*** methods manipulate are assumed private
- The ***set*** method starts with the prefix "**set**" and the rest of the method name is the name of data or field
- ***set*** methods are declared public and void, accepting a parameter of the same type as that of the fields they change
- The ***get*** method starts with the prefix "**get**" and the rest of the method name is the name of data or field
- ***get*** methods are no-arg methods that are declared public and that return a value of the same type as that of the fields they access

set and *get* methods

```
public class OurDate {
   private int year;
   private int month;
   private int day;

   public OurDate(){
     year=2010;
     month=1;
    day=1;
   }
```

set and *get* methods

```
public OurDate(int y,int m,int d){
    year    =  y;
    month = m;
    day     = d;
   }
public void setYear(int yr){
          this.year = yr;
        }
public int getYear(){
       return this.year;
  }
```

set and *get* methods

```
public void setMonth(int mth){
    this.month = mth;
   }
public int getMonth(){
    return this.month;
   }
} //end OurDate class
```

Summary

- *SecureRandom* (package java.security) can be used to generate nondeterministic random numbers
- In Java, all arguments are passed by value
- Method-call conversion occurs when passing a value of one type as an argument to a method that expect a different type
- You can have methods in a class with the same name as long the methods sharing the same name have distinct signatures

Material Revision

Updated by Cyr M. Bakinde, Sept 2017