# 6.2 Static Methods and Static Variables

- OutLine
  - » Static Variables
  - » Static Methods
  - » Dividing the Task of a `main` Method into Subtasks
  - » Adding a `main` Method to a class
  - » The `Math` Class
  - » Wrapper Classes

# Static Methods

- Some methods don't not need an [redacted] to do their job
  - » For example, methods to calculate area: just pass the required parameters and return the area
- Use the [redacted] instead of an object name to invoke them
- For example
  - » `CircleFirstTry` is a class with methods to perform calculations on circles: `CircleFirstTry.area(myRadius);`
  - » Notice that the the method invocation uses "*`ClassName.`*" instead of " *`circleObject.`*"
- Also called [redacted] ***methods//Static methods***

# Static Methods

- Declare static methods with the *static* modifier, for example:

```
public static double area(double radius)
   ...
```

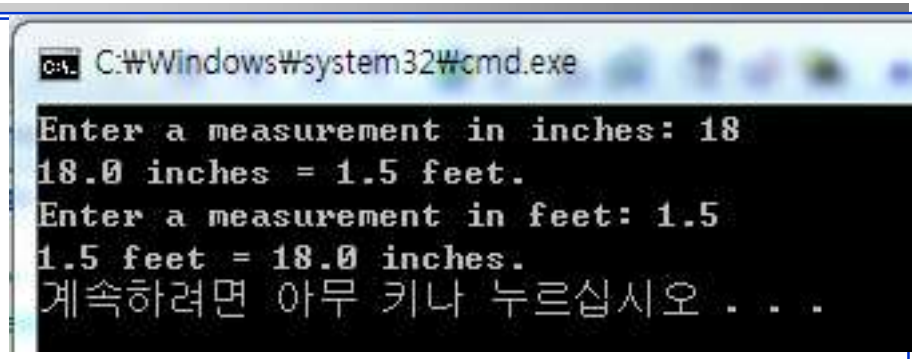
- Since a static method doesn't need a calling object, it cannot refer to a (nonstatic) instance variable of the class.

# List 6.5 Static Methods

```
/**
 Class of static methods to perform dimension conversions.
*/
public class DimensionConverter
{
        // A static constant ; it could be private here
        public static final int INCHES_PER_FOOT = 12;

        public static double convertFeetToInches(double feet)
        {
                return feet * INCHES_PER_FOOT;
        }

        public static double convertInchesToFeet(double inches)
        {
                return inches / INCHES_PER_FOOT;
        }
}
```

# List 6.6 Using Static Methods

```
import java.util.Scanner;

/**
 Demonstration of
  using the class DimensionConverter.
*/
public class DimensionConverterDemo
{
        public static void main(String[] args)
        {
                Scanner keyboard = new Scanner(System.in);
                System.out.print("Enter a measurement in inches: ");
                double inches = keyboard.nextDouble();
                double feet = DimensionConverter.convertInchesToFeet(inches);
                System.out.println(inches + " inches = " + feet + " feet.");

                System.out.print("Enter a measurement in feet: ");
                feet = keyboard.nextDouble();
                inches = DimensionConverter.convertFeetToInches(feet);
                System.out.println(feet + " feet = " + inches + " inches.");
        }
}
```



```
// Listing 6.7

import java.util.Scanner;
/**
Class with static and nonstatic members.
*/
public class SavingsAccount
{
    private double balance;  // instance variable
    public static double interestRate = 0;  // static variable
    public static int numberOfAccounts = 0;
    // A nonstatic method can reference a static variable
    public SavingsAccount ()
    {
        balance = 0; // instance variable
        numberOfAccounts++; // static variable
    }
    // A static method can reference a static variable but not an instance variable
    public static void setInterestRate (double newRate)
    {
        interestRate = newRate;  // interestRate  : static variable
    }
```

```
public static double getInterestRate ()
{
    return interestRate;
}
public static double getNumberOfAccounts ()
{
    return numberOfAccounts;
}
public void deposit (double amount)
{
    balance = balance + amount;
}
public double withdraw (double amount)
{
    if (balance >= amount)
        balance = balance - amount;
    else
        amount = 0;
    return amount;
}
```

```
    public void addInterest ()
    {
        double interest = balance * interestRate;
        // you can replace interestRate with getInterestRate()
        balance = balance + interest;
    }

    public double getBalance ()
    {
        return balance;
    }

    // A static method cannot call a nonstatic method
    //                    unless it has an object to do so
    public static void showBalance (SavingsAccount account)
    {
        System.out.print (account.getBalance ()); // public double getBalance ()
    }
}
```

**// Listing 6.7 (Modified-Error!!!)**

```
import java.util.Scanner;
/**
Class with static and nonstatic members.
*/
public class SavingsAccount
{
    private double balance;
    public static double interestRate = 0;
    public static int numberOfAccounts = 0;

    public SavingsAccount ()
    {
        balance = 0;
        numberOfAccounts++;
    }


    public static void setInterestRate (double newRate)
    {
        interestRate = balance; // balance is a instance variable
        //        interestRate = newRate;
    }
```

**// Listing 6.7 (Modified)**

D:\@@강의-200901-3-자바 프로그래밍\@강의안\Chap06\savingAccount\SavingsAccount.java:20:

```
        interestRate = balance;
                     ^
1 error

Tool completed with exit code
```

```
//    public static void setInterestRate (double newRate)
//    {
//        interestRate = balance;
//    }
    public static void setInterestRate (SavingsAccount account)
    {
        interestRate =
    }
    public double getBalance ()
    {
        return balance;
    }
```

**// Listing 6.7 (Modified)**

```
 public void addInterest ()
   {
       double interest = balance * interestRate;
       // you can replace interestRate with getInterestRate()
       balance = balance + interest;
   }


   public double getBalance ()
   {
       return balance;
   }


   public static void showBalance (SavingsAccount account)
   {
 //      System.out.print (account.getBalance ());
        System.out.print (getBalance ());
   }
}
```

**// Listing 6.7 (Modified)**

```
D:\@@강의-200901-3-자바
프로그래밍\@강의안\Chap06\savingAccount\SsavingsAccount.java:69:

        System.out.print (getBalance ());
                    ^
1 error

Tool completed with exit code 1
```

```
// Listing 6.8
// Using Static and Nonstatic methods

public class SavingsAccountDemo
{
    public static void main (String [] args)
    {
        SavingsAccount.setInterestRate (0.01);
        SavingsAccount mySavings = new SavingsAccount ();
        SavingsAccount yourSavings = new SavingsAccount ();
        System.out.println ("I deposited $10.75.");
        mySavings.deposit (10.75);
        System.out.println ("You deposited $75.");
        yourSavings.deposit (75.00);
        System.out.println ("You deposited $55.");
        yourSavings.deposit (55.00);
        double cash = yourSavings.withdraw (15.00);
        System.out.println ("You withdrew $" + cash + ".");
```

```
        if (yourSavings.getBalance () > 100.00)
        {
            System.out.println ("You received interest.");
            yourSavings.addInterest ();
        }
        System.out.println ("Your savings is $" +
                yourSavings.getBalance ());
        System.out.print ("My savings is $");
        SavingsAccount.showBalance (mySavings);
        System.out.println ();
        int count = SavingsAccount.getNumberOfAccounts ();
        System.out.println ("We opened " + count
                " savings accounts today.");
    }
}
```

C:₩WINDOWS₩system32₩cmd.exe

```
I deposited $10.75.
You deposited $75.
You deposited $55.
You withdrew $15.0.
You received interest.
Your savings is $116.15
My savings is $10.75
We opened 2.0 savings accounts today.
계속하려면 아무 키나 누르십시오 . . .
```

# *Java Tip*: You Can Put a **`main`** in Any Class

- Usually **`main`** is by itself in a class definition.
- Sometimes it makes sense to have a **`main`** method in a regular class definition.
- When the class is used to create objects, **the `main` method is ignored.**
- Adding a diagnostic **`main`** method to a class makes it easier to test the class's methods.
- Because main must be static, you can't invoke nonstatic methods of the class in **`main`** unless you create an object of the class.

  오브젝트를 사용해야하는 이유

- Normally you wouldn't put a **`main`** method in a class that is used to create objects unless it is for test purposes.

# DISPLAY Placing a main method in a Class Definition- PlayCircle2.java

```
// DISPLAY Placing a main method in a Class Definition
public class PlayCircle2
{   public static final double PI = 3.14159;
    private double diameter;
    public static void main(String[] args)
    {       /* Because main must be static,
                you can't invoke nonstatic methods of the class in main
                unless you create an object of the class.  */
        PlayCircle2 circle = new PlayCircle2( );
        circle.setDiameter(2); //public void setDiameter(double newDiameter)
        System.out.println("If circle has diameter 2,");
        circle.showArea( ); //public void showArea( )
        System.out.println("Now, you choose the diameter:");
        PlayCircle2.areaDialog( );  // static method
        // Because this main is inside the definition of the class PlayCircle2,
        //      you can omit this PlayCircle2, if you wish.
    }
```

```
public void setDiameter(double newDiameter)
  {
     diameter = newDiameter;
  }
  public static double area(double radius)
  {
     return (PI*radius*radius);
  }
  public void showArea( )
  {
     System.out.println("Area is " + area(diameter/2));
  }
  public static void areaDialog( )
  {
     Scanner keyboard = new Scanner(System.in);
     System.out.println("Enter the diameter of a circle:");
     double newDiameter = keyboard.nextDouble( );
     PlayCircle c = new PlayCircle( );
     c.setDiameter(newDiameter);
     c.showArea( );
  }
}
```

# Static Variables

- Static variables are [redacted] by all objects of a class
  - » Variables declared `static final` are considered constants – value cannot be [redacted]

- Variables declared `static` (without [redacted]) can be changed
  - » Only [redacted] instance of the variable exists
  - » It can be accessed by all instances of the class

# Static Variables

- Static variables also called [redacted] *variables*
  - » Contrast with [redacted] *variables*
- Do not confuse class variables with variables of a class type
- Both static variables and instance variables are sometimes called [redacted] or [redacted]

# Static Variables(= [redacted] variable)

- The `StaticDemo` program in the text uses a static variable:

  ```
  private static int numberOfInvocations = 0;
  ```

- Similar to definition of a named constant, which is a special case of static variables.
- May be public or private but are usually private for the same reasons instance variables are.
- [redacted] and it can be accessed by any object of the class.
- May be initialized (as in example above) or not.
- Can be used to let objects of the [redacted] class coordinate.
- Not used in the rest of the text.

# DISPLAY A Static Variable - StaticDemo.java

```
// DISPLAY A Static Variable
public class StaticDemo
{// object1 and object2 use the same static variable numberOfInvocations
   private static int numberOfInvocations = 0;
   public static void main(String[] args)
   {
      int i;
      StaticDemo object1 = new StaticDemo( );
      for (i = 1; i <=10 ; i++)
        object1.outPutCountOfInvocations( );   // 10 invocations
      StaticDemo object2 = new StaticDemo( );
      for (i = 1; i <=10 ; i++)
        object2.justADemoMethod( );  // 10 invocations
      System.out.println("Total number of invocations = "
                    + numberSoFar( ));// one more increment (static method)
   }
```

```
public void justADemoMethod( )
  {
     numberOfInvocations++;
     //In a real example, more code would go here.
  }

  public void outPutCountOfInvocations( )
  {
     numberOfInvocations++;
     System.out.println(numberOfInvocations);
  }

  public static int numberSoFar( )
  {
     numberOfInvocations++;   // one more increment
     return numberOfInvocations;
  }
}
```

```
C:\WINDOWS\system32\cmd.exe
1
2
3
4
5
6
7
8
9
10
Total number of invocations
계속하려면 아무 키나 누르십시오 . . .
```

# DISPLAY - StaticPrivate

```
// DISPLAY  A Static Variable     //????
public class StaticPrivateDemo0
{// object1 and object2 use the same static variable numberOfInvocations
   private int numberOfInvocations = 0;   // static ==> private
   public static void main(String[] args)
   {
      int i;
      StaticDemo object1 = new StaticDemo( );
      for (i = 1; i <=10 ; i++)
         object1.outPutCountOfInvocations( );   // 10 invocations

      StaticDemo object2 = new StaticDemo( );
      for (i = 1; i <=10 ; i++)
         object2.justADemoMethod( );  // 10 invocations

      System.out.println("Total number of invocations = "
                    + object2.numberSoFar( ));// one more increment
   }
```

```
public void justADemoMethod( )
  {
     numberOfInvocations++;
     System.out.println(numberOfInvocations);
     //In a real example, more code would go here.
  }

  public void outPutCountOfInvocations( )
  {
     numberOfInvocations++;
     System.out.println(numberOfInvocations);
  }

  public  int numberSoFar( )   // static ==> private
  {
     numberOfInvocations++;   // one more increment
     return numberOfInvocations;
  }
}
```

C:₩WINDOWS₩system32₩cmd.exe

```
1
2
3
4
5
6
7
8
9
10
Total number of invocations
계속하려면 아무 키나 누르십시오 . . .
```

# DISPLAY 5.8 - StaticPrivate

```
// DISPLAY 5.8 A Static Variable
public class StaticPrivateDemo
{// object1 and object2 use the same static variable numberOfInvocations
   private int numberOfInvocations = 0;   // static ==> private

   public static void main(String[] args)
   {
      int i;
      StaticPrivateDemo object1 = new StaticPrivateDemo( );
      for (i = 1; i <=10 ; i++)
        object1.outPutCountOfInvocations( );   // 10 invocations
      StaticPrivateDemo object2 = new StaticPrivateDemo( );
      for (i = 1; i <=10 ; i++)
        object2.justADemoMethod( );  // 10 invocations
      System.out.println("Total number of invocations = "
                    + object2.numberSoFar( ));// one more increment (static
method)
   }
```

C:₩WINDOWS₩system32₩cmd.exe

```
1
2
3
4
5
6
7
8
9
10
1
2
3
4
5
6
7
8
9
10
Total number of invocations
계속하려면 아무 키나 누르십시오 . . .
```

**// Listing 6.9 A Main Method with Repetitive Code**

```
public class SpeciesEqualsDemo
{
    public static void main (String [] args)
    {
        Species s1 = new Species (), s2 = new Species ();
        s1.setSpecies ("Klingon Ox", 10, 15);
        s2.setSpecies ("Klingon Ox", 10, 15);
        System.out.println ("Now change one Klingon Ox.");
        s2.setSpecies ("klingon ox", 10, 15); //Use lowercase

    if (s1 == s2)
        System.out.println ("Match with ==.");
    else
        System.out.println ("Do Not match with ==.");
```

```
if (s1.equals (s2))
    System.out.println ("Match with the method equals.");
else
    System.out.println ("Do Not match with the method equals.");

System.out.println ("Now change one Klingon Ox.");
s2.setSpecies ("klingon ox", 10, 15); //Use lowercase

if (s1.equals (s2))
    System.out.println ("Match with the method equals.");
else
    System.out.println ("Do Not match with the method equals.");
    }
}
```

C:₩WINDOWS₩system32₩cmd.exe

```
Now change one Klingon Ox.
Do Not match with ==.
Match with the method equals.
Now change one Klingon Ox.
Match with the method equals.
계속하려면 아무 키나 누르십시오 . . .
```

```
// Listing 6.10

public class SpeciesEqualsDemo
{
   public static void main (String [] args)
   {
      Species s1 = new Species (), s2 = new Species ();
      s1.setSpecies ("Klingon Ox", 10, 15);
      s2.setSpecies ("Klingon Ox", 10, 15);
      testEqualsOperator (s1, s2);
      testEqualsMethod (s1, s2);
      System.out.println ("Now change one Klingon Ox.");
      s2.setSpecies ("klingon ox", 10, 15); //Use lowercase
      testEqualsMethod (s1, s2);
   }
```

→ object 없이 바로 메소드 사용

```
    private static void testEqualsOperator (Species s1, Species s2)
    {
        if (s1 == s2)
            System.out.println ("Match with ==.");
        else
            System.out.println ("Do Not match with ==.");
    }

    private static void testEqualsMethod (Species s1, Species s2)
    {
        if (s1.equals (s2))
            System.out.println ("Match with the method equals.");
        else
            System.out.println ("Do Not match with the method equals.");
    }
}
```

**// Listing 6.11  Placing a main Method in a Class Definition**

```
import java.util.Scanner;
public class Species
{
    private String name;
    private int population;
    private double growthRate;

    public void readInput ()
    {
        Scanner keyboard = new Scanner (System.in);
        System.out.println ("What is the species' name?");
        name = keyboard.nextLine ();
        System.out.println (
                "What is the population of the species?");
        population = keyboard.nextInt ();
```

```
    while (population < 0)
    {
        System.out.println ("Population cannot be negative.");
        System.out.println ("Reenter population:");
        population = keyboard.nextInt ();
    }
    System.out.println (
            "Enter growth rate (% increase per year):");
    growthRate = keyboard.nextDouble ();
}


public void writeOutput ()
{
    System.out.println ("Name = " + name);
    System.out.println ("Population = " + population);
    System.out.println ("Growth rate = " + growthRate + "%");
}
```

```
/**
Precondition: years is a nonnegative number.
Returns the projected population of the receiving object
after the specified number of years.
*/
public int predictPopulation (int years)
{
    int result = 0;
    double populationAmount = population;
    int count = years;
    while ((count > 0) && (populationAmount > 0))
    {
        populationAmount = (populationAmount +
                (growthRate / 100) * populationAmount);
        count -- ;
    }
    if (populationAmount > 0)
        result = (int) populationAmount;
    return result;
}
```

```
public void setSpecies (String newName, int newPopulation,
        double newGrowthRate)
{
    name = newName;
    if (newPopulation >= 0)
        population = newPopulation;
    else
    {
        System.out.println ("ERROR: using a negative population.");
        System.exit (0);
    }
    growthRate = newGrowthRate;
}


public String getName ()
{
    return name;
}
```

```
public int getPopulation ()
{
    return population;
}


public double getGrowthRate ()
{
    return growthRate;
}


public boolean equals (Species otherObject)
{
    return (name.equalsIgnoreCase (otherObject.name)) &&
        (population == otherObject.population) &&
        (growthRate == otherObject.growthRate);
}
```

```
public static void main (String [] args)
{
    Species speciesToday = new Species ();
    System.out.println ("Enter data on today's species:");
    speciesToday.readInput ();
    speciesToday.writeOutput ();
    System.out.println ("Enter number of years to project:");
    Scanner keyboard = new Scanner (System.in);
    int numberOfYears = keyboard.nextInt ();
    int futurePopulation =
        speciesToday.predictPopulation (numberOfYears);
    System.out.println ("In " + numberOfYears +
            " years the population will be " +
            futurePopulation);
    speciesToday.setSpecies ("Klingon ox", 10, 15);
    System.out.println ("The new species is:");
    speciesToday.writeOutput ();
}
}
```

C:₩WINDOWS₩system32₩cmd.exe

```
Enter data on today's species:
What is the species' name?
Hosik
What is the population of the species?
10
Enter growth rate (% increase per year):
15
Name = Hosik
Population = 10
Growth rate = 15.0%
Enter number of years to project:
100
In 100 years the population will be 11743134
The new species is:
Name = Klingon ox
Population = 10
Growth rate = 15.0%
계속하려면 아무 키나 누르십시오 . . .
```

# The **Math** Class

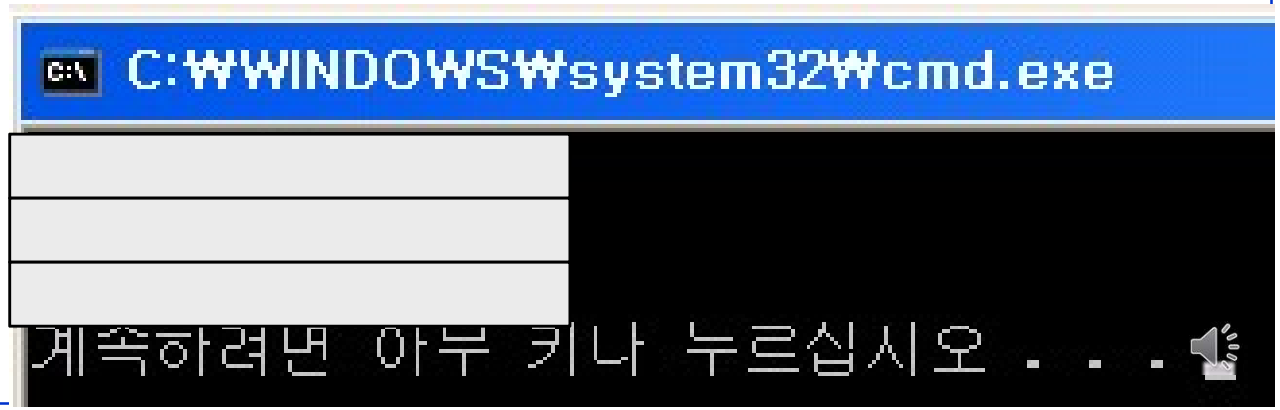
```
// Math Test     ?????
public class MathTest
{
    public static void main(String[] args)
    {
        double start = 3.56;
        int answer = (int) Math.round(start);
        int lowerAnswer = (int) Math.floor(start);
        int highAnswer= (int) Math.ceil(start);

        System.out.println("answer= " + answer);
        System.out.println("lowerAnswer=  "+lowerAnswer);
        System.out.println("highAnswer=  "+highAnswer);
    }
}
```

static 메소드

반올림

올림

버림



# The **`Math`** Class

- Math Constants : Includes constants `Math.PI` (approximately 3.14159) and `Math.E` (base of natural logarithms which is approximately 2.72)
- Includes three similar static methods: `round`, `floor`, and `ceil`
  - » All three return whole numbers (although they are type `double`)
  - » **`Math.round`** returns the whole number nearest its argument

`Math.round(3.3)` returns `3.0` and `Math.round(3.7)` returns `4.0`

  - » **`Math.floor`** returns the nearest whole number that is equal to or less than its argument

`Math.floor(3.3)` returns `3.0` and `Math.floor(3.7)` returns `3.0`

  - » **`Math.ceil`** (short for ceiling) returns the nearest whole number that is equal to or greater than its argument

`Math.ceil(3.3)` returns `4.0` and `Math.ceil(3.7)` returns `4.0`

# Figure 6.3

| Name | Description | Argument Type | Return Type | Example | Value Returned |
|---|---|---|---|---|---|
| pow | Power | double | double | Math.pow(2.0,3.0) | 8.0 |
| abs | Absolute value | int, long, float, or double | Same as the type of the argument | Math.abs(-7)<br>Math.abs(7)<br>Math.abs(-3.5) | 7<br>7<br>3.5 |
| max | Maximum | int, long, float, or double | Same as the type of the arguments | Math.max(5, 6)<br>Math.max(5.5, 5.3) | 6<br>5.5 |
| min | Minimum | int, long, float, or double | Same as the type of the arguments | Math.min(5, 6)<br>Math.min(5.5, 5.3) | 5<br>5.3 |
| round | Rounding | float or double | int or long, respectively | Math.round(6.2)<br>Math.round(6.8) | 6<br>7 |
| ceil | Ceiling | double | double | Math.ceil(3.2)<br>Math.ceil(3.9) | 4.0<br>4.0 |
| floor | Floor | double | double | Math.floor(3.2)<br>Math.floor(3.9) | 3.0<br>3.0 |
| sqrt | Square root | double | double | sqrt(4.0) | 2.0 |

# DISPLAY Circle.java, CircleDemo2.java

```
// DISPLAY  Predefined Constants
/**
 Class with static methods to perform calculations on circles.
*/
public class Circle
{
   public static double area(double radius)
   {
      return (Math.PI*radius*radius);  // Math.PI
   }

   public static double circumference(double radius)
   {
      return (Math.PI*(radius + radius));// Math.PI
   }
}
```

```
// DISPLAY 5.10. Predefined COnstantsv
public class CircleDemo2
{
   public static void main(String[] args)
   {
      double radius;

      System.out.println("Enter the radius of a circle in inches:");
      radius = SavitchIn.readLineDouble( );
      System.out.println("A circle of radius "
                              + radius + " inches");
      System.out.println("has an area of " +
         Circle.area(radius) + " square inches,");
      System.out.println("and a circumference of " +
         Circle.circumference(radius) + " inches.");
   }
}
```

[redacted]

- Used to wrap [redacted] types in [redacted] structure
- All primitive types have an equivalent (Wrapper) class
- The class includes **useful** [redacted] **and static** [redacted] including one to convert back to the primitive type

| Primitive type | Class type | Method to convert back |
|---|---|---|
| `int` | `Integer` | `intValue()` |
| `long` | `Long` | `longValue()` |
| `float` | `Float` | `floatValue()` |
| `double` | `Double` | `doubleValue()` |
| `char` | `Character` | `charValue()` |

# Wrapper class example: **`Integer`**

- Declare an `Integer` class variable:

```
Integer n = new Integer(42);
```

- Convert the value of an `Integer` variable to its primitive type, `int`:

```
int i = n.intValue();//intValue returns
  an int
```

- Some useful `Integer` constants:
  - » **`Integer.MAX_VALUE`** - the maximum integer value the computer can represent
  - » **`Integer.MIN_VALUE`** - the smallest integer value the computer can represent

# Wrapper class example: `Integer`

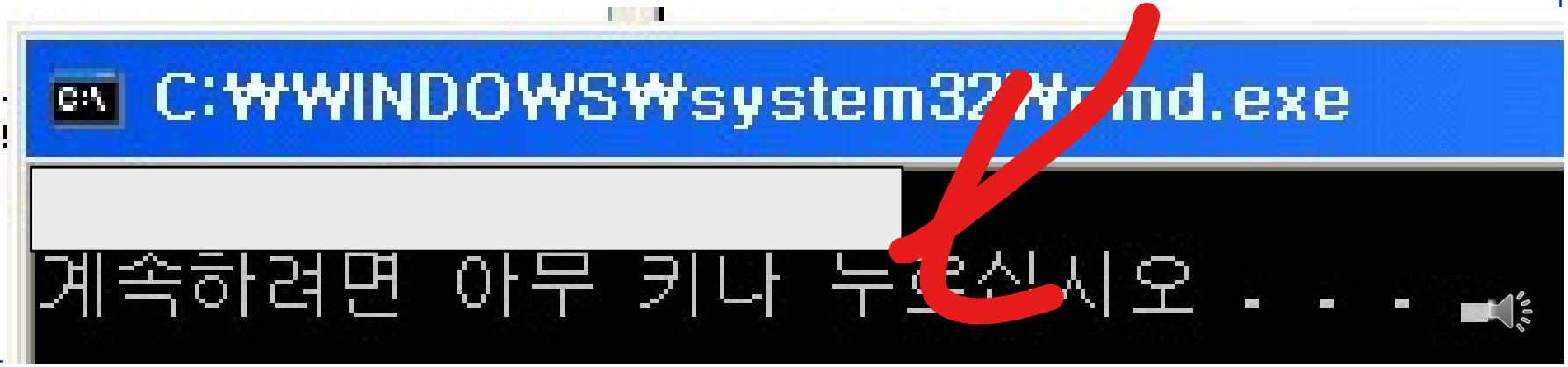
- Some useful `Integer` methods:
  - » **`Integer.valueOf("123")`** to convert a string of numerals to an integer
  - » **`Integer.toString(123)`** to convert an `Integer` to a `String`
- The other wrapper classes have similar constants and methods : parseDouble(), parseInt()
  - » Double.parseDouble("199.98")
  - » Double.parseDouble(theString.trim())
    - – trim() : trims off leading and trailing whitespace, such as blanks
- See the text for useful methods for the class `Character`

# Figure 6.4 Static methods in class `Character`

| Name | Description | Argument Type | Return Type | Examples | Return Value |
|---|---|---|---|---|---|
| `toUpperCase` | Convert to uppercase | `char` | `char` | `Character.toUpperCase('a')`<br>`Character.toUpperCase('A')` | `'A'`<br>`'A'` |
| `toLowerCase` | Convert to lowercase | `char` | `char` | `Character.toLowerCase('a')`<br>`Character.toLowerCase('A')` | `'a'`<br>`'a'` |
| `isUpperCase` | Test for uppercase | `char` | `boolean` | `Character.isUpperCase('A')`<br>`Character.isUpperCase('a')` | `true`<br>`false` |
| `isLowerCase` | Test for lowercase | `char` | `boolean` | `Character.isLowerCase('A')`<br>`Character.isLowerCase('a')` | `false`<br>`true` |
| `isLetter` | Test for a letter | `char` | `boolean` | `Character.isLetter('A')`<br>`Character.isLetter('%')` | `true`<br>`false` |
| `isDigit` | Test for a digit | `char` | `boolean` | `Character.isDigit('5')`<br>`Character.isDigit('A')` | `true`<br>`false` |
| `isWhitespace` | Test for whitespace | `char` | `boolean` | `Character.isWhitespace(' ')`<br>`Character.isWhitespace('A')` | `true`<br>`false` |

Whitespace characters are those that print as white space, such as the blank, the tab character (`'\t'`), and the line-break character (`'\n'`).

```
// Wrapper Test

public class CharWrapperTest
{
    public static void main(String[] args)
    {
        Character c1 = new Character('a');
        Character c2 = new Character('A');

        if (c1.equals(c2))
            System.out.println(c1.charValue() + "is the same as " +
c2.charValue());
                else
            System.out.println(c1.charValue() + "is not the same as " +
c2.charValue());

    }
}
```



# Usage of wrapper classes

차이점

There are some important differences in the code to use wrapper classes and that for the primitive types

Wrapper Class

- variables contain the *address* of the value
- variable declaration example:
  `Integer n;`
- variable declaration & init:
  `Integer n = new Integer(0);`
- assignment:
  `n = new Integer(5);`

Primitive Type

- variables contain the value
- variable declaration example:
  `int n;`
- variable declaration & init.:
  `int n = 0;`
- assignment:
  `n = 99;`

# Two personalities of Wrapper Class

- 1) to produce objects of the class Integer
  - » Integer n = new Integer(42);

- 2) serves as a library of useful static methods
  - » int number = Integer.parseInt(inputString);

끝