MORE ABOUT OBJECTS AND METHODS

Chapter 6

Objectives

- Define and use constructors
- Write and use static variables and methods
- Write and use overloaded methods

CONSTRUCTORS

Ch 6.1

Constructors: Outline

- Defining Constructors
- Calling Methods from Constructors
- Calling a Constructor from Other Constructors

Defining Constructors

- A constructor
  - is a special method called automatically when an instance of an object is created with new

```
ClassName x = new ClassName();
```

  - has the same name as the class name.
  - can have parameters to specify initial values if desired
  - but cannot return values and it is not a void method
  - May have multiple definitions
    - Each with different numbers or types of parameters

- A class contains **at least one** constructor.

Defining Constructors

- Example class to represent rectangles

Rectangle
- width: int - height: int - color: String
+ setWidth (int w): void + setHeight (int h): void + setColor (String c): void + getWidth(): int + getHeight(): int + getColor(): String

```java
public class RectangleTest {
    public static void main(String[] args) {
        Rectangle box1 = new Rectangle();
        box1.display();
    }
}
```

```java
public class Rectangle {
    private int width;
    private int height;
    private String color;
    public Rectangle() {
        width = 1;
        height = 1;
        color = "white";
    }
    public void setWidth(int w){  width = w;}
    public void setHeight(int h){ height = h; }
    public void setColor(String c){ color = c; }
    public int getWidth() { return width; }
    public int getHeight() { return height; }
    public String getColor() { return color; }
    public void display() {
    System.out.println("Width= "+width+", Height= "+height+", Color= "+
    color);}}
```

Default constructor

Sample screen output

```
Width= 1, Height= 1, Color= white
```

Defining Constructors

- Constructor without parameters is the default constructor
- Java will define this automatically, but only if the class designer does not define any constructors
- If you do define a constructor, Java will not automatically define a default constructor, but you can still add one.
- Usually default constructors are not included in class diagrams (UML)

```java
public class Rectangle {
    private int width;
    private int height;
    private String color;
    public Rectangle() {
        width = 1;
        height = 1;
        color = "white";}
    public Rectangle(int w, int h, String c) {
        width = w;
        height = h;
        color = c;}
    public Rectangle(int w, int h) {
        width = w;
        height = h; }// rest of methods
}
```

Don't forget to check the validity of received values

```java
public class RectangleTest {
    public static void main(String[] args) {
        Rectangle box1 = new Rectangle(5, 10 , "Black");
        box1.display();
        Rectangle box2 = new Rectangle(3, 20);
        box2.display();}}
```

Sample screen output

```
Width= 5, Height= 10, Color= Black
Width= 3, Height= 20, Color= null
```

Calling Methods from Other Constructors

```
public class Rectangle {
    private int width;
    private int height;
    private String color;
    public Rectangle() {
        width = 1;
        height = 1;
        color = "white";}
    public Rectangle(int w, int h, String c) {
        setWidth(w);
        setHeight(h);
        setColor(c);
    }// rest of methods
}
```

Checking validity of received values can be done in setters methods

```
public class RectangleTest {
    public static void main(String[] args) {
        Rectangle box1 = new Rectangle(5, 10, "Black");
        box1.display();}}
```

Sample screen output

```
Width= 5, Height= 10, Color= Black
```

```java
public class Rectangle {
    private int width;
    private int height;
    private String color;
    public Rectangle() {
        width = 1;
        height = 1;
        color = "white";
    }
    public Rectangle(int w, int h,
    String c) {
        setWidth(w);
        setHeight(h);
        setColor(c);
    }
    // rest of methods
}
```

You cannot use an existing object to call a constructor

```java
public class RectangleTest {
    public static void main(String[] args) {
        Rectangle box1 = new Rectangle();
        box1.Rectangle(5, 10, "Black");
    }
}
```

Compilation error

To change the instance values of an object after it has been created, you should call one of the set methods.

```java
public class Rectangle {
    private int width;
    private int height;
    private String color;

    public Rectangle(int w, int h,
    String c) {
        setWidth(w);
        setHeight(h);
        setColor(c);
    }
    // rest of methods
}
```

Compilation error: constructor Rectangle() is undefined!

```java
public class RectangleTest {
    public static void main(String[] args) {
        Rectangle box1 = new Rectangle();
        box1.display();
    }
}
```

If you <u>do</u> define a constructor, Java will <u>not</u> automatically define a default constructor

Constructors & set methods calling a private method

```
public class Rectangle {
    private int width;
    private int height;
    private String color;

    public Rectangle(int w, int h, String c) { set(w, h, c); }
    public Rectangle(int w, int h) { set(w, h, "White"); }
    public Rectangle(String c) { set(1, 1, c); }
    public Rectangle() { set(1,1, "White"); }

    public void setWidth(int w){ set(w, height, color);}
    public void setHeight(int h) { set(width, h, color); }
    public void setColor(String c){ set(width, height, c); }
    private void set(int w, int h, String c) {
        width = w;
        height = h;
        color = c;}
        // rest of methods
}
```

Calling Constructor from Other Constructors

```
public class Rectangle {
    private int width;
    private int height;
    private String color;
    public Rectangle(int w, int h,
    String c) {
        width = w;
        height = h;
        color = c;
    }
    public Rectangle(int w, int h) {
        this(w, h, "White"); }
    public Rectangle(String c) {
        this(1, 1, c); }
    public Rectangle() {
        this(1,1, "White"); }
    // rest of methods
}
```

- **In the other constructors use the this reference to call initial constructor**
- **Constructor call must be the first statement in a constructor**

Copy Constructor

- Sometimes we want to create an exact copy (duplicate) of an existing object, such that the changes made in this copy does not reflect on the original object.
- **Copy constructor** is special type of constructor
  - Takes an existing object of the same class as parameter.
  - Copies *each field* (attribute) of the existing object into the new object.

Example: Constructor with No-Parameter

```
public class Person
{
String name;
int age;
// Constructor
public Person(){
    name = " ";
    age = 0;}
}
```

```
Person P1;

P1    = new Person( )    ;
```

A

B

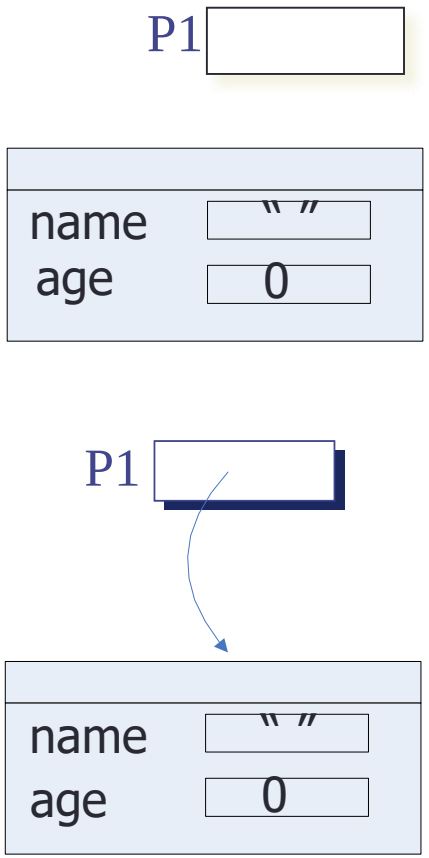
C

Code

A. The instance variable is allocated in memory.

B. The object is created with initial state

C. The reference of the object created in B is assigned to the variable.



State of Memory

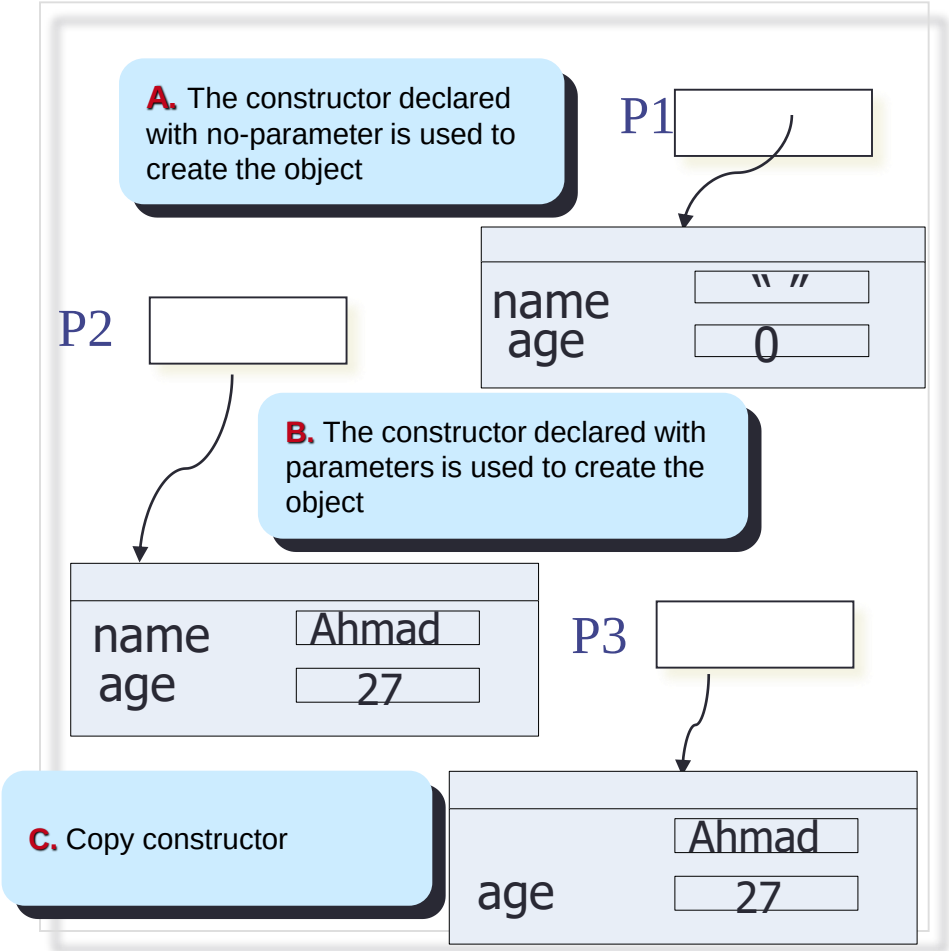
Example: Class with Multiple Constructors

```
Public class Person
{
String name;
int age;
// Constructor
public Person(){
    name = " ";
    age = 0;}
public Person(Person other)
{   name = other.name;
    age = other.age; }
public Person(String n, int a )
{   name = n;
    age = a; }
}
```

```
Person P1 , P2, P3;

P1   =  new Person()

P2   = new Person("Ahmed",27);

P3   = new Person(P2);
```

A B C

Code



State of Memory

ACTIVITY: CLASS EXAMPLE

Example - Account Class

- Create a Java class based on the following UML :

Account
- number :int - balance : double
+deposit (double amount) : void +withdraw(double amount) : void

- The class should:
  - Have a **default constructor** that initializes the attributes to default values, and **another constructor** that initializes the data attributes to given values, and a **copy constructor**.
  - Method **deposit** will add to balance
  - Method **withdraw** will reduce the balance
  - Provide set() and get() methods for each attribute.

- In the main() method of the class TestAccount write statements that will call both constructors and test class Accounts capabilities.

Class Account

```
public class Account
{ // definition of attributes (data)
 private int number;
 private double balance;
 // constructor
 public Account ()
 {  number=0;
    balance=0; }
 public Account (int n , double b)
 {  number=n;
    balance=b; }
 public Account (Account a)
 {  number= a.number;
    balance= a.balance; }

 // definition of operations
 public void deposit (double amount)
 {  balance = balance + amount;
 } //end of deposit
```

```
public void withdraw(double
amount)
 {  if (balance >= amount)
       balance = balance - amount;
 } //end of withdraw
public void setNumber (int n)
{  number = n;
} //end of setNumber

public void setBalance (double b)
{  balance=b;
} //end of setBalance

public int getNumber()
{  return  number ;
} //end of getNumber

public double getBalance() {
   return balance;
} //end of getBalance } //end of
class
```

Class TestAccount

```
public class TestAccount
  {
 public static void main(String[] args) {

Account Account1=new Account();
Account Account2=new Account(1,6200);
Account Account3=new Account(Account2);
Account1. setNumber (2) ;
Account1. setBalance (4300) ;
Account2. deposit (550) ;
Account1. withdraw(200);
Account3. deposit (50)
System.out.println(Account1.getBalance()+ "-" +Account2.getBalance()+
"-" + Account3.getBalance() );

}
      }
```

STATIC VARIABLES AND METHODS

Ch 6.2

Static Variables & Methods: Outline

- Static Variables
- Static Methods
- Dividing the Task of a `main` Method into Subtasks
- Adding a `main` Method to a class (optional)
- Predefined methods
- Wrapper Classes (optional)

Static Variables

- They are variables declared as `static`
- They are shared by all objects of a class
  - Only one instance of the variable exists
  - It can be accessed by all instances of the class via the class name or the object name
- Static variables are also called class variables
  - Contrast with instance variables
  - Note: Do not confuse class variables with variables of a class type
- Both static (class) variables and instance variables are sometimes called fields or data members or attributes
- Underline static variables in UML diagram

Static Variables

- **Values** of variables declared:
  - `static final` cannot be changed, they are constants
  - `static` (without `final`) can be changed

- A common examples of static attributes is to have a variable that keeps track of how many objects of a class have been created.

Constructor and Static attribute

```
public class Person {

  String name;
  int age;
  public static int numofPerson=0;

  // Constructor
  public Person()
  {  name = " ";
     age = 0;
     numofPerson++; }

  public Person(String n , int a)
  {  name = n;
     age = a;
     numofPerson++; }

} // end class Person
```

```
...
public static void
 main(String[]  args)
{
  System.out.println
    (Person.numofPerson);

  Person P1 = new Person();
  Person P2 = new
        Person("ahmed", 27);

   System.out.println
    (Person.numofPerson);

}
...
```

Constructor and Static attribute

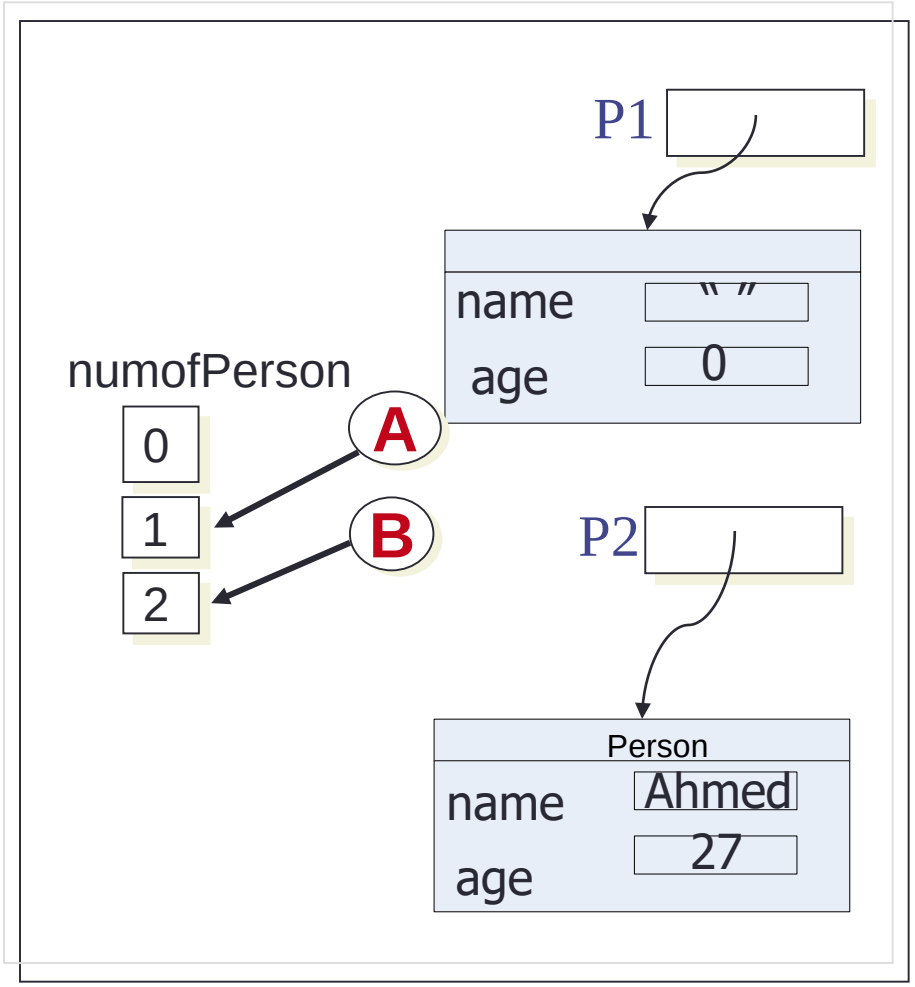
```
public static void main(string[]
  args)
{
  Person P1 = new Person();
  Person P2 = new Person ("ahmed",
  27);
}
```

```
Person P1 , P2;

P1  =  new Person()

P2   = new Person("Ahmed",27);
```

A

B

Code



State of Memory

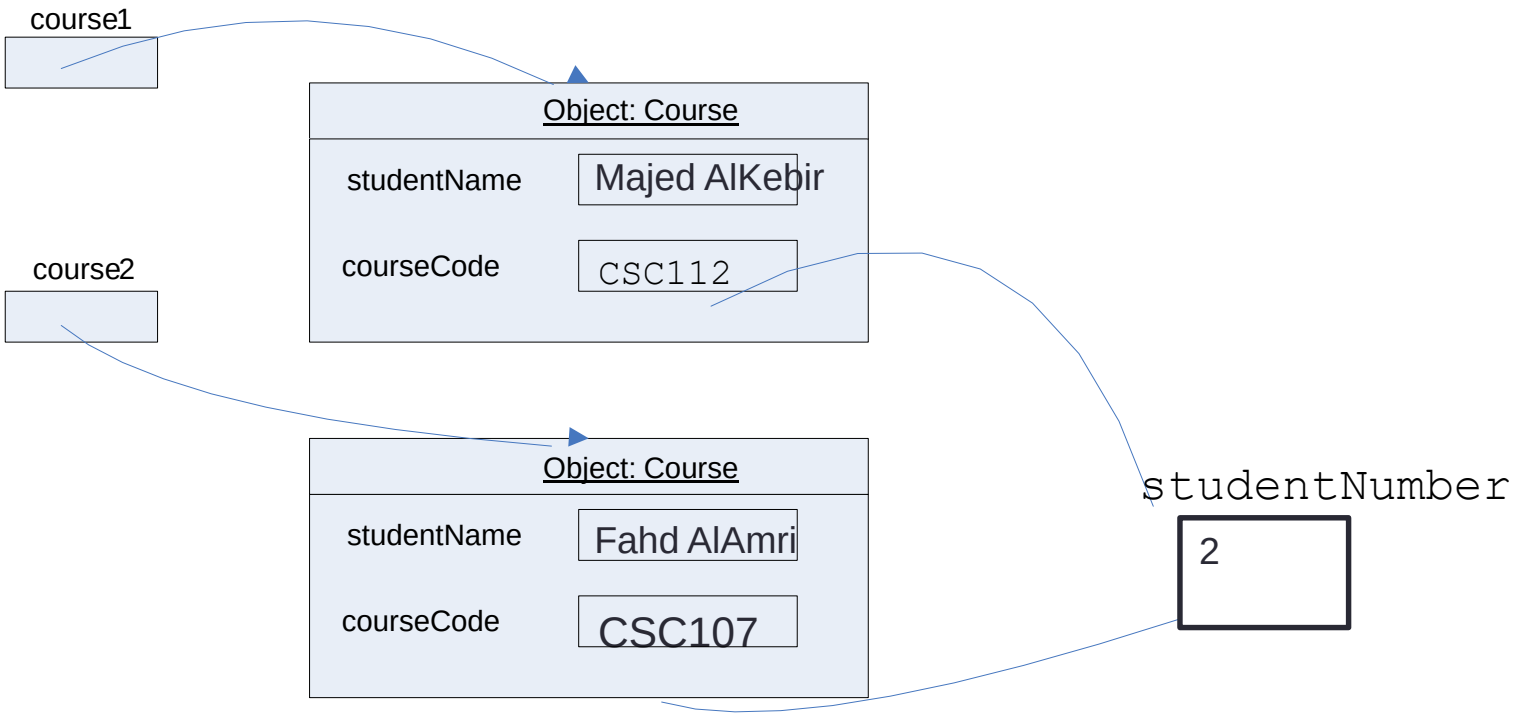
```java
class Course {
    // attributes
    public String studentName; // Instance variables
    public String courseCode ; // Instance variables
    public static int studentNumber; // Class variables

}
```

Course
+studentName: String +courseCode: String +studentNumber:int

```java
public class CourseRegistration {
    public static void main(String[] args) {
        Course course1, course2;
        //Create and assign values to course1
        course1 = new Course( ); Course.studentNumber = 1;
        course1.courseCode= “CSC112”;
        course1.studentName= “Majed AlKebir“;
        //Create and assign values to course2
        course2 = new Course( ); Course.studentNumber ++;
        course2.courseCode= “CSC107”;
        course2.studentName= “Fahd AlAmri”;
        System.out.println(course1.studentName + " has the course “+
                           course1.courseCode + “ ” + course1.studentNumber);
        System.out.println(course2.studentName + " has the course “+
                           course2.courseCode + “ ” + course2.studentNumber);
    }
}
```

CourseRegistration
+main()

Class Attributes and instance Attributes



Static Methods

- Some methods may have no relation to any type of object
- Example
  - Compute max of two integers
  - Convert character from upper- to lower case

- Static methods declared <u>in</u> a class
  - Can be invoked <u>without</u> using an object
  - Instead use the class name

- For example, Math library functions.

Static M

LISTING 6.5 Sta

```
/**
 Class of static m
*/
public class Dimen
{
    public static

    public static
    {
        return fee
    }

    public static
    {
        return inc
    }
}
```

LISTING 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.println("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.");
```

```
Enter a measurement in inches: 18
18.0 inches = 1.5 feet.
Enter a measurement in feet: 1.5
1.5 feet = 18.0 inches.
```

Sample screen output

Mixing Static and Nonstatic Methods

LISTING 6.7 Mixing Static and Non-static Members in a Class *(part 1 of 2)*

```
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 = newRate;
    }

    public static double getInterestRate()
    {
        return interestRate;
    }
```

An instance variable (nonstatic)

Static variables

A nonstatic method can reference a static variable.

A static method can reference a static variable but not an instance variable.

Mixing Static and Nonstatic Methods

```
public static int getNumberOfAccounts()
{
    return numberOfAccounts;
}

public void deposit(double amount)
{
    balance = balance + amount;
}

public double withdraw(do
{
    if (balance >= amount
        balance = balance
    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;
    }

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

A nonstatic method can reference a static variable or call a static method.

A static method cannot call a nonstatic method unless it has an object to do so.

Mixing Static and Nonstatic Methods

LISTING 6.8 Using Static and Non-static 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(15.75);
        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.");
    }
}
```

Screen Output

```
I deposited $10.75.
You deposited $75.
You deposited $55.
You withdrew $15.75.
You received interest.
Your savings is $115.3925
My savings is $10.75
We opened 2 savings accounts today.
```

Tasks of `main` in Subtasks

- Program may have
  - Complicated logic
  - Repetitive code
- Create static methods to accomplish subtasks
- Consider example code, listing 6.9
  a `main` method with repetitive code
- Note alternative code, listing 6.10
  uses helping methods

Tasks of `main` in Subtasks

```
public class RectangleTest {
    public static void main(String[] args) {
    Rectangle box1 = new Rectangle(5, 10 , "Black");
    Rectangle box2 = new Rectangle(5, 10, "Red");

    if(box1.equals(box2))
        System.out.println("Math with equals method");
    else
        System.out.println("Do not match with equals method");

    // change the color of box 2 to Black
    box2.setColor("Black");

    if(box1.equals(box2))
        System.out.println("Math with equals method");
    else
        System.out.println("Do not match with equals method");
    }
}
```

Repetitive code

Tasks of `main` in Subtasks

```
public class RectangleTest {
    public static void main(String[] args) {
    Rectangle box1 = new Rectangle(5, 10 , "Black");
    Rectangle box2 = new Rectangle(5, 10, "Red");

    testEqualsMethod(box1, box2);
    // change the color of box 2 to Black
    box2.setColor("Black");
    testEqualsMethod(box1, box2);

    }
    private static void testEqualsMethod(Rectangle r1, Rectangle r2) {
        if(r1.equals(r2))
                System.out.println("Math with equals method");
        else
                System.out.println("Do not match with equals method");
    }
}
```

Calling one static method

Sample screen output

```
Do not match with equals method
Math with equals method
```

Adding Method main to a Class

- Method main used so far in its own class within a separate file
- Often useful to include method main within class definition
  - To create objects in other classes
  - To be run as a program

Adding Method main to a Class

```
public class Rectangle {
   private int width;
   private int height;
   private String color;

   // the rest of Rectangle methods here

   public static void main(String[] args) {
      Rectangle box1 = new Rectangle(5, 10 , "Black");
      box1.display();
    }
}
```

Remember static methods can't reference non-static methods (display(), for example) unless it has an object to do so.