Parameter passing: call-by-value

```
public class CBV
{
    public static void main(String[] args)
    {
      int i=9;
      System.out.println("Initially : " + i);
      changeI(i);
      System.out.println("Finally   : " + i);
    }

    private static void changeI(int ii)
    {
      ii=ii*3;
      System.out.println("In changeI: " + ii);
    }
}
```

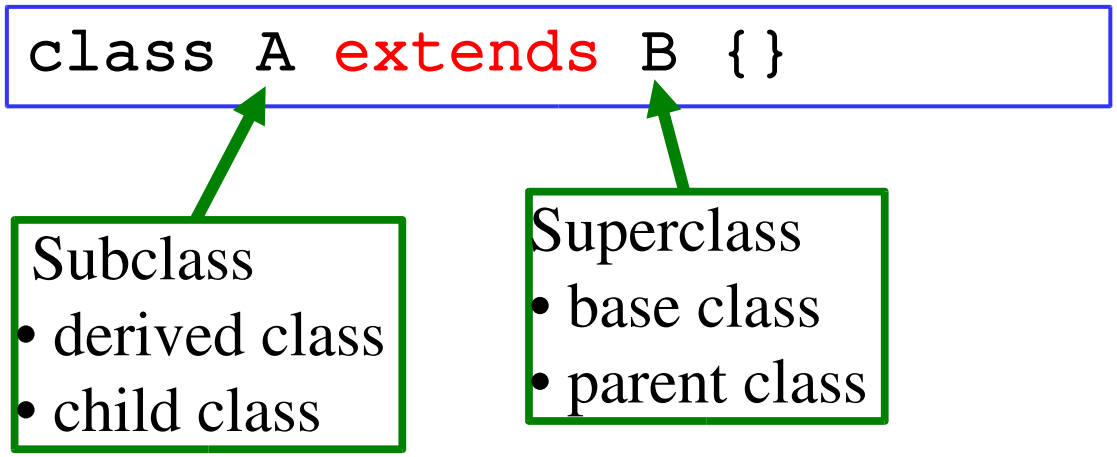
Passing references to objects (arrays)

```
public class CBR
{
   public static void main(String[] args)
   {
     int[] i={9};
     System.out.println("Initially : " + i[0]);
     changeI(i);
     System.out.println("Finally   : " + i[0]);
   }

   private static void changeI(int[] ii)
   {
     ii[0]=ii[0]*3;
     System.out.println("In changeI: " + ii[0]);
   }
}
```

Inheritance

- A way to reuse classes so that new classes may be designed without starting from scratch
  - methods and fields are “inherited” and need not be repeated
  - method overriding



Subclass

- `A` is a subclass of `B`.
  - `public class A extends B ...`
- All methods and instance variables of `B` are automatically inherited by `A`.
- `A` may have additional methods and variables of its own.
- `A` may in turn be the superclass for some new class C

```
public class Person {
    private String name="";
    private int age=0;
    public void setName(String s) {
       name=s; }
    public void setAge(int a) {
       age=a; }
    public String getName() {
       return name;
    }
    public int getAge() {
       return age;
    }
    public String toString() {
       return "name="+name+
                      "age="+age;
    }
}
```

```
public class Adult extends Person {
    private Person spouse=null;
    public void marry(Person p) {
       if (spouse != null) {
          spouse=p;
          p.marry(this);
       }
    }
    public void divorce() {
       if (spouse!=null) {
             spouse.divorce();
             spouse=null;
          }
    }
}
```

Accessor Methods

- Variables should never be public!!!!
  - except static that never change
- Always use “accessor methods”
- Good practice to even use them inside class
- Allows change to data representation without any code rewrite

Access Privileges

There are 4 levels of access for class methods and variables

- private
  - access
    - only within the class in which it is declared
  - Use
    - DANGER!!! -- use this rarely
    - variables and methods for **internal** consumption
- protected
  - access
    - inside the class in which it is declared
    - all extensions of that class
  - Use
    - frequent – perhaps the most common access level
    - variables and methods for the class and all subclasses

access ... continued

- “” -- otherwise known as “package”
  - access
    - inside the class in which it is declared
    - all extensions of that class
    - all classes in the same package
  - Use
    - never!!
- “public”
  - access
    - to everyone
  - use
    - accessor methods
    - other methods
    - unchanging static variables

```
public class Person1 {
   protected String name="";
   protected int age=0;
   public Person1(String s) {
      name=s; }
   public void setAge(int a) {
      age=a; }
   public String getName() {
      return name;
   }
   public int getAge() {
      return age;
   }
   public String toString() {
      return "name="+name+
                    "age="+age;
   }
}
```

```
public class Adult1 extends Person1 {
   protected Adult1 spouse=null;
   public Adult1(String n) { super(n); }
   public String toString() {
      if (spouse==null)
         return "name="+name+ "--single";
      else
         return "name="+name + " married to " + spouse.g
   }
   public void marry(Adult1 p) {
      if (spouse == null) {
         spouse=p;
         p.marry(this);
      }}
   private boolean divorcing=false;
   public void divorce() {
      divorcing=true;
      if (spouse!=null && !divorcing) {
            spouse.divorce();
            spouse=null;
         }
      divorcing=false;
   }}
```

Overriding

- Instead of simple inheritance, super class's methods can be redefined in a subclass
- Overridden methods keep the exact same name, type and exceptions thrown.
  - e.g., toString method of Person1 is overridden by toString method of Adult1
- If name were kept the same but type changed, i.e. number and type of the method's parameters, it is called overloading

Overloading

```
public class Bigamist1 extends Adult1
{
   protected Adult1 spouse2=null;
   public Bigamist1(String n) { super(n); }
   public void marry(Adult1 p1, Adult1 p2) {
      marry(p1);
      if (spouse2 == null) {
         spouse2=p2;
         p2.marry(this);
      }
   }
   public void divorce(Adult1 p) {
      if (spouse==p) {
            spouse.divorce();
            spouse=null;
            if (spouse2!=null) {
               spouse=spouse2;
               spouse2=null;
            }}
      if (spouse2==p) {
         spouse2.divorce();
         spouse2=null;
      }}
```

```
   public String toString()
   {
      String rtn="name="+getName();
      if (spouse==null)
         return rtn + " Single";
      rtn = rtn + " married to " + spouse.getName();
      if (spouse2==null)
         return rtn;
      return rtn + " and " +spouse2.getName();
   }
   public static void main(String[] args)
   {
      Bigamist1 m = new Bigamist1("Dick");
      Adult1 w = new Adult1("Jane");
      Adult1 w2 = new Adult1("Mary");
      System.out.println(m);
      m.marry(w);
      System.out.println(m);
      System.out.println(w);
      m.divorce();
      System.out.println(m);
      m.marry(w, w2);
      System.out.println(m);
   }}
```

Constructors

- Every class comes with a default constructor with no parameters
- Additional constructors with any number of parameters can be defined.
  - doing so turns off the default constructor -- DANGER
- If a subclass has a custom constructor
  - A constructor of the superclass is called
  - Then statements in the subclass' constructor are executed
- A subclass may explicitly access its super class's constructors through the key word `super`

Example -- Constructors

```
public class A
{
   String s="???";
   public String toString()
   { return s; }
   public static void main(String[] args)
   {
      A a = new A();
      System.out.println(a);
   }}
```

```
public class B
{
   String s="???";
   public B(String ss) { s=ss; }
   public String toString()
   { return s; }
   public static void main(String[] args)
   {
      B b = new B();
      System.out.println(b);
   }
}
```

What is the result of A and B?

More Constructors

```
public class B
{
   String s="???";
   public B(String ss) { s=ss; }
   public String toString()
   { return s; }
   public static void main(String[] args)
   {
      B b = new B("q");
      System.out.println(b);
   }
}
```

```
public class C extends B {
   public static void main(String[] args) {
      C c = new C();
   }
}
```

Problem: C does not compile

error message:

symbol : constructor B()

location: class B

Interpretation:

C is looking for no arg constructor in B

Fix:

1. Put "public B() { }" in B
2. Put "public C() { super(null); } in C

The granddaddy class

- In Java, a class may extend only one class (single inheritance/single parenting)
- As we have seen, a class can have many children. This leads to a tree structure.
- At the top of the inheritance tree is the class `Object` – all classes extend `Object` by default
  - `class A extends Object`

What you inherit from `Object`

- Methods in `Object` meant to be overridden
  - `toString()`
  - `getClass()`
  - `equals()`
  - `clone()`
  - `wait()`
  - `notify()`

References and casting

- Reference variable of type superclass can be used to refer to a subclass or siblings, but not vice versa

```
Person p = new Person();
Attorney a = new Attorney();
Knight k = new Knight();
a = p; // not ok
p = a; //ok
a = k; //ok
```

- If must be done, use explicit casting

```
a = (Attorney) p; //dangerous
```

- Can be tested using `instanceof`