Methods (2A)

Young Won Lim
12/10/14

Please send corrections (or suggestions) to youngwlim@hotmail.com.

This document was produced by using OpenOffice.

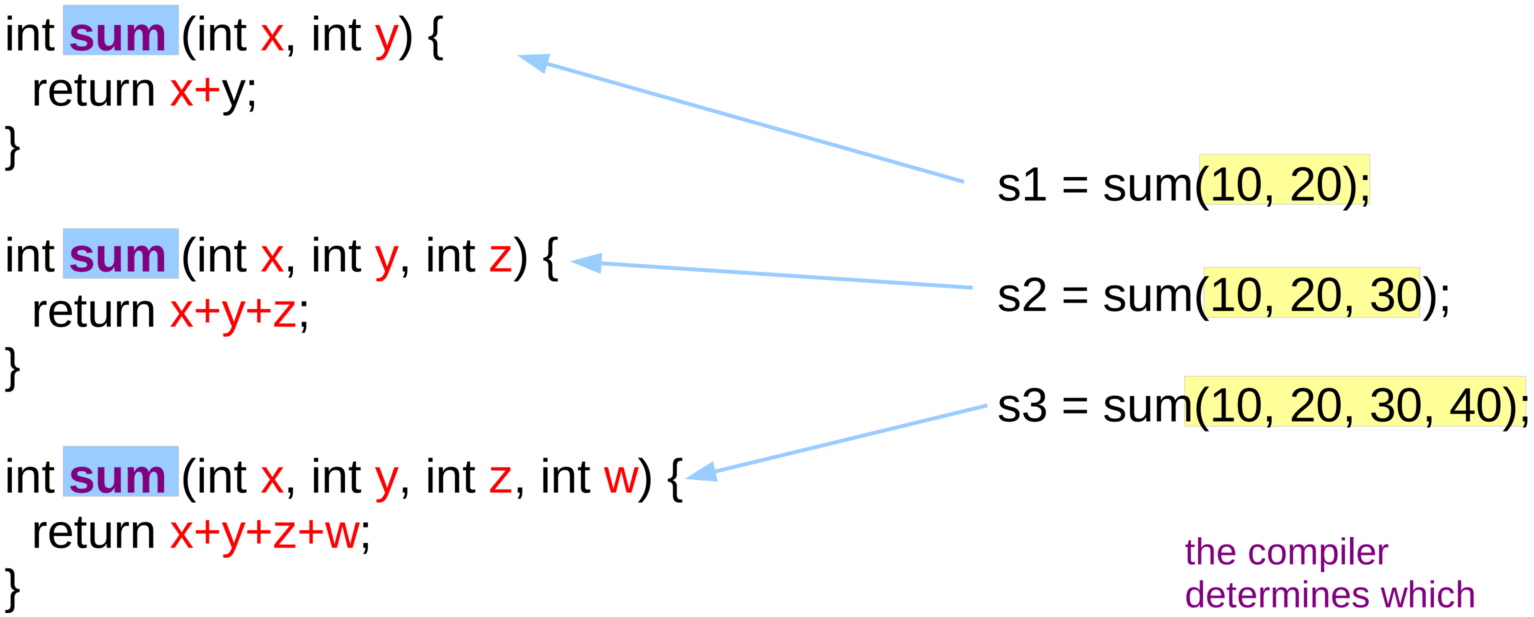
Function Overloading

C <math.h>	C++ <cmath>
int **abs** (int n);	int **abs** (int n);
long int **labs** (long int n);	long int **abs** (long int n);
double **fabs** (double x);	double **fabs** (double x);

the same function name

different function prototypes

Method Overloading



```
int sum (int x, int y) {
  return x+y;
}

int sum (int x, int y, int z) {
  return x+y+z;
}

int sum (int x, int y, int z, int w) {
  return x+y+z+w;
}
```

```
s1 = sum(10, 20);

s2 = sum(10, 20, 30);

s3 = sum(10, 20, 30, 40);
```

the compiler determines which function is called

the same function name

different function prototypes

Constructor Functions

```
class Ccircle {
  public int r;

  public Ccircle ()       { r = 1; }
  public Ccircle (int x) { r = x; }

  public void      setR (int x)  { r = x; }
  public int         getR ()        { return r; }
  public double area ()          { return
                                          3.14*r*r;  }
}
```

the constructor function name:
the same as the **class name**

<u>no return type</u>; not even <u>void</u>

automatically called whenever a new object of this class is created

used for initialization purpose

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

  Ccircle C1 = new Ccircle();
  Ccircle C2 = new Ccircle(10);


}
```

The **default constructor** is without any parameter.

the **default constructor** must be declared in addition to any other constructors defined

Overloaded Constructor Functions

```
class Ccircle {
  public int r;

  public Ccircle ()       { r = 1; }
  public Ccircle (int x) { r = x; }

  public void     setR (int x)  { r = x; }
  public int        getR ()         { return r; }
  public double area ()         { return
                                             3.14*r*r;  }
}
```

the same function name

different function prototypes

No Friend Functions

```
public class A {
    private int p = 31415;

    public class Meth {
        public int getP() { return p; }
        // no public constructor
        private Meth() { }
    }   // inner class  Meth

   public void giveKeyTo(B other) {
        other.receiveKey(new Meth());
    }
}
```

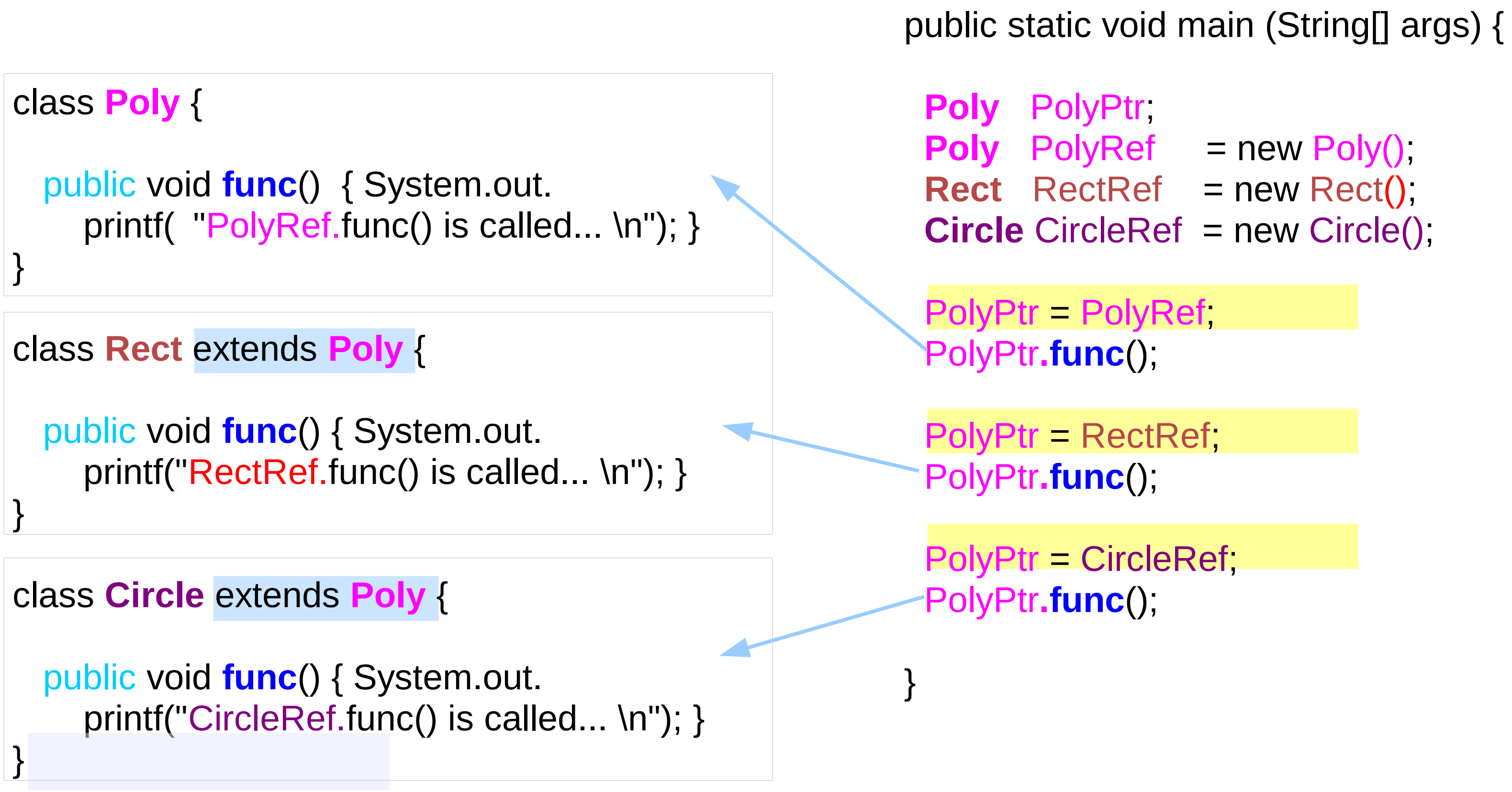
The object of **B** cannot access any fields (p) or methods (getP()) of an object of **A**.

After a.giveKeyTo(this) creates the object of the inner class Meth, the class **B** can access. This a.giveKeyTo() method requires a valid **B** as an argument. Only the object of **B** can get access.

```
public class B {
    private A.Meth key;

    public void receiveKey(A.Meth key) {
        this.key = key;
    }

    public void usageExample() {
        A a = new A();

        // int foo = a.p;
        // doesn't work, not accessible

        a.giveKeyTo(this);
        int m = key.getP();
        System.out.println(m);
    }
}
```

http://stackoverflow.com/questions/14226228/implementation-of-friend-concept-in-java

(Virtual) Method Overriding



```
class Poly {

   public void func()  { System.out.
      printf(  "PolyRef.func() is called... \n"); }
}
```

```
class Rect extends Poly {

   public void func() { System.out.
      printf("RectRef.func() is called... \n"); }
}
```

```
class Circle extends Poly {

   public void func() { System.out.
      printf("CircleRef.func() is called... \n"); }
}
```

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

  Poly   PolyPtr;
  Poly   PolyRef     = new Poly();
  Rect   RectRef    = new Rect();
  Circle CircleRef  = new Circle();

  PolyPtr = PolyRef;
  PolyPtr.func();

  PolyPtr = RectRef;
  PolyPtr.func();

  PolyPtr = CircleRef;
  PolyPtr.func();

}
```

all non-static methods are by default "virtual functions."

*methods with the **final** keyword cannot be overridden*

***private** methods cannot be inherited and are non-virtual.*

Abstract Methods – Pure Virtual Member Functions

```
abstract class Poly {

   public int m;
   abstract void func() ;
}
```

```
class Rect extends Poly {

   public void func() { System.out.
     printf("Rect func() is called... \n"); }
}
```

```
class Circle extends Poly {

   public void func() { System.out.
    printf("Circle func() is called... \n"); }
}
```

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

  Poly   PolyRef ;
  Rect   RectRef    = new  Rect();
  Circle CircleRef  = new  Circle();

  ~~PolyRef = new Poly();~~
  ~~PolyRef.func();~~

  PolyRef = RectRef;
  PolyRef.func();

  PolyRef = CircleRef;
  PolyRef.func();

}
```

Classes containing pure virtual functions are termed "**abstract**";
they cannot be instantiated directly.

Interface – Pure Virtual Member Functions

```
interface Poly {

   void func() ;

}
```

```
class Rect implements Poly {

   public void func() { System.out.
     printf("Rect func() is called... \n"); }
}
```

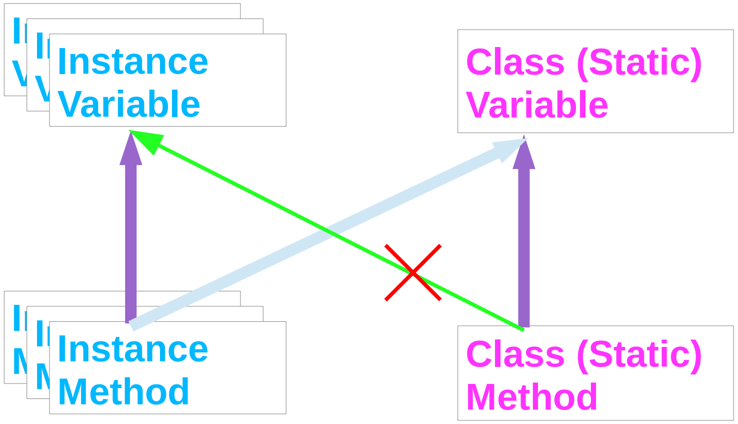
```
class Circle implements Poly {

   public void func() { System.out.
    printf("Circle func() is called... \n"); }
}
```

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

  Poly   PolyRef ;
  Rect   RectRef    = new  Rect();
  Circle CircleRef  = new  Circle();

  ~~PolyRef = new Poly();~~
  ~~PolyRef.func();~~

  PolyRef = RectRef;
  PolyRef.func();

  PolyRef = CircleRef;
  PolyRef.func();

}
```

Inferface consists only **abstract** methods

Static Methods vs Instance Methods

```
public class CC {

  int ii;
  void ifun() { System.out.
        println(“ii= “ + ii);
  }

  static int ci;
  static void cfun() { System.out.
        println(“ci= “ + ci);
  }

}
```



o.ci
o.cfun()

Object Reference

CC.ci
CC.cfun()

Class Name

CC o = new CC();

Static Methods Example (1)

```
public class CC {

  int ii;
  void ifun() { System.out.
        println(“ii= “ + ii);
  }

  static int ci;
  static void cfun() { System.out.
        println(“ci= “ + ci);
  }

}
```

```
public class Test  {

  int ii;
  void ifun() { System.out.
        println(“ii= “ + ii);
  }

  static int ci;
  static void cfun() { System.out.
        println(“ci= “ + ci);
  }

  public static void main(String[] args) {
        ifun();✗
        cfun();

        CC o = new CC();

        o.ifun();
        o.cfun();✗

        CC.cfun();

  }

}
```

Static Methods Example (2)

```
class CRect {
  public int r;
  public static int s;

  // CRect () { s = 0; }

  static void func() { System.out
        printf("static s=%d\n", s++);
  }
}
```

constructor cannot initialize static members

```
public class test {
    int ~~CRect.s~~ = 0;

    public static int main(void) {
      CRect Cobj;
      CRect.s = 0;

      CRect.func();
      CRect.func();

      CRect.func();

       return 0;
    }
}
```

Not working

OK

```
public class test {

  public static int prLine(void) {
     System.out.println(“============”);
  }

  public static int main(void) {
     prLine();
     return 0;
  }
}
```

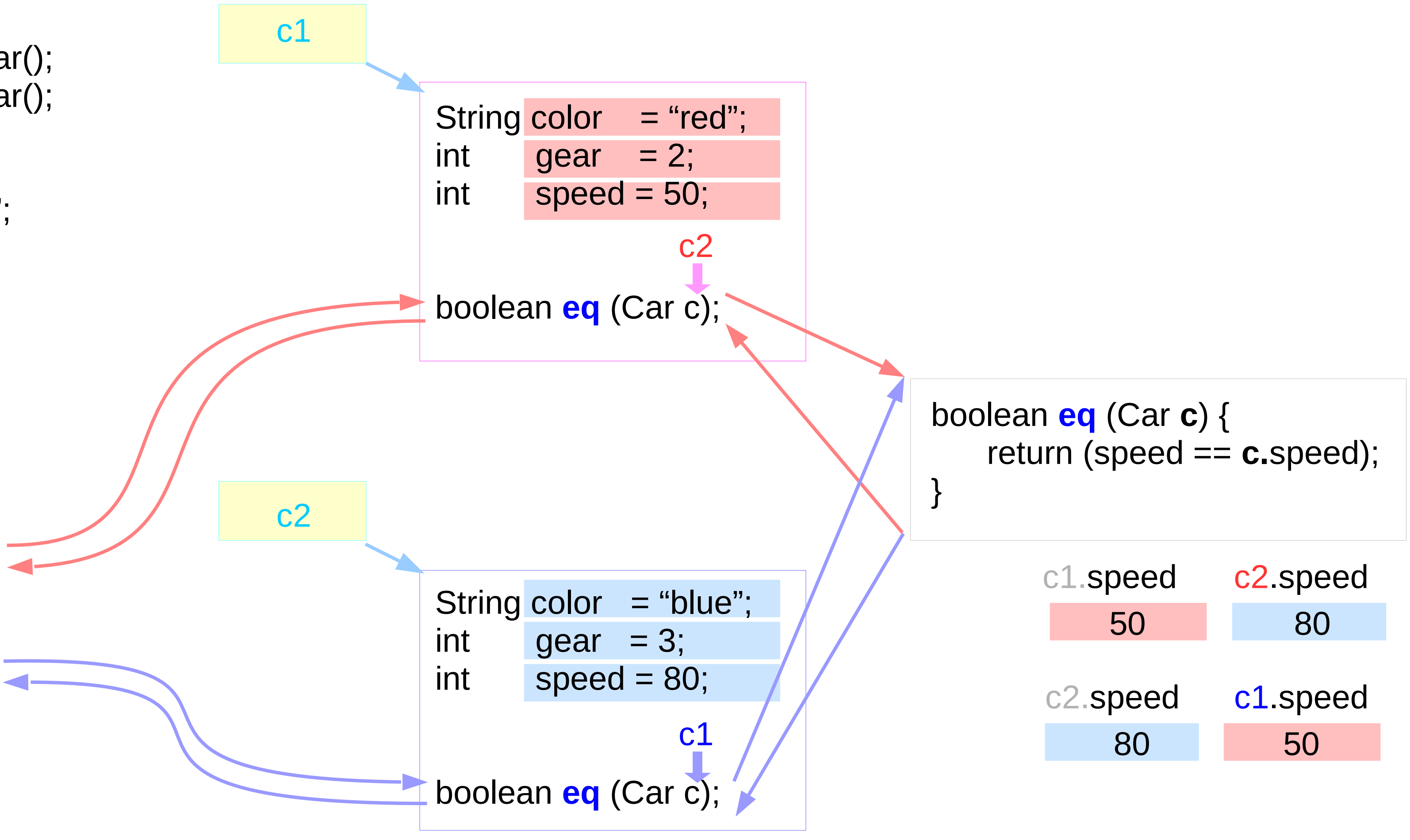
Instance Method Call Diagram (1)

```
class Car {
   String color;
   int       gear;
   int       speed;

   boolean eq (Car c) {
        return (speed == c.speed);
   }
}
```

```
class Test {

   public static void main (String[] args) {

      Car c1 = new Car();
      Car c2 = new Car();

      c1.color = “red”;
      c2.color = “blue”;

      c1.gear = 2;
      c2.gear = 3;

      c1.speed = 50;
      c2.speed = 80;


      c1.eq(c2);

      c2.eq(c1);

  }

}
```

Instance Method Call Diagram (2)

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

    Car c1 = new Car();
    Car c2 = new Car();

    c1.color = “red”;
    c2.color = “blue”;

    c1.gear = 2;
    c2.gear = 3;

    c1.speed = 50;
    c2.speed = 80;


    c1.eq(c2);


    c2.eq(c1);

}
```



```
String color   = “red”;
int    gear    = 2;
int    speed = 50;

boolean eq (Car c);
```

```
String color   = “blue”;
int    gear   = 3;
int    speed = 80;

boolean eq (Car c);
```

```
boolean eq (Car c) {
    return (speed == c.speed);
}
```

c1.speed	c2.speed
50	80
c2.speed	c1.speed
80	50

Static Method Call Diagram

```
class CRect {
  public int r;
  public static int s;

  // CRect () { s = 0; }

  static void func() { System.out
         printf("static s=%d\n", s++);
  }
}
```

constructor cannot initialize static members

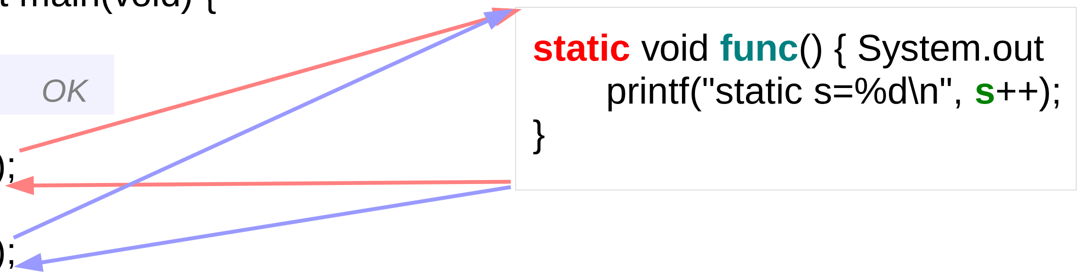
```
public class test {
    int ~~CRect.s~~ = 0;

    public static int main(void) {
      CRect Cobj;
      CRect.s = 0;

      CRect.func();

      CRect.func();

        return 0;
    }
}
```

Not working

OK

static int s;

0



```
static void func() { System.out
       printf("static s=%d\n", s++);
}
```

References

[1] W Savitch, “Absolute C++”

[2] P.S. Wang, “Standard C++ with objected-oriented programming”

[3] http://www.cplusplus.com