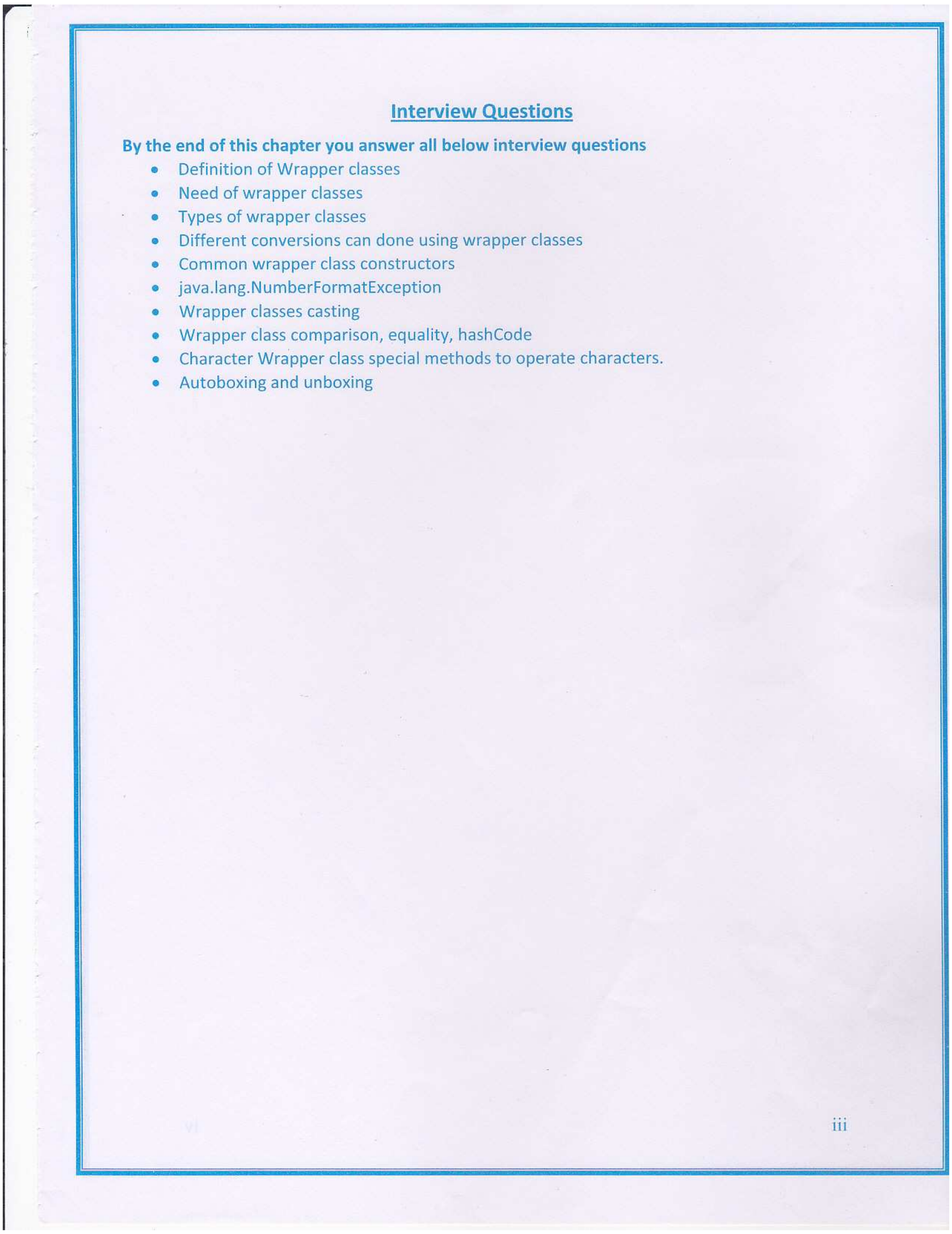
Chapter 5

Wrapper Classes
with Auto Boxing and Unboxing

- In this chapter, You will learn
  - Definition and need of Wrapper Classes
  - Types of Wrapper Classes
  - Different types of conversions
  - *java.lang.NumberFormatException*
  - Special case with Boolean class
  - Character class special methods
  - Casting in wrapper classes
  - Wrapper Classes comparison
  - Auto Boxing and Unboxing
- By the end of this chapter- you will be comfortable in working with wrapper classes.



Interview Questions

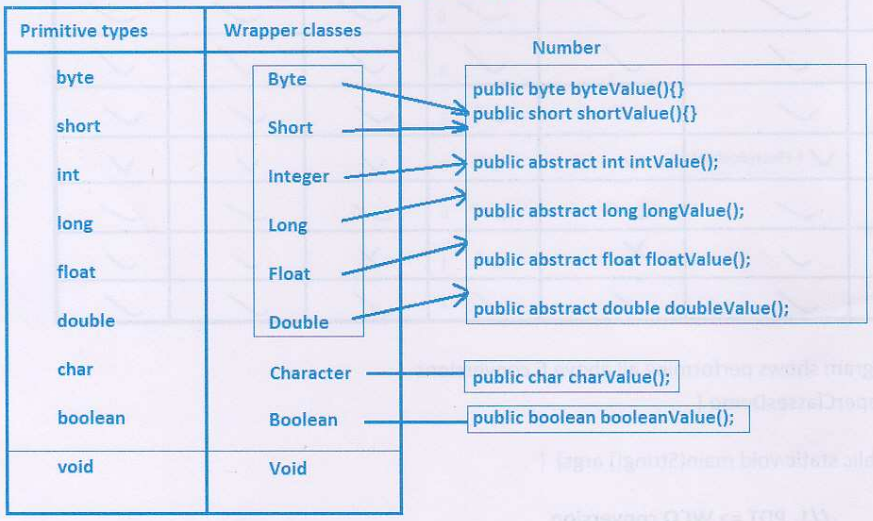
By the end of this chapter you answer all below interview questions

- Definition of Wrapper classes
- Need of wrapper classes
- Types of wrapper classes
- Different conversions can done using wrapper classes
- Common wrapper class constructors
- java.lang.NumberFormatException
- Wrapper classes casting
- Wrapper class comparison, equality, hashCode
- Character Wrapper class special methods to operate characters.
- Autoboxing and unboxing

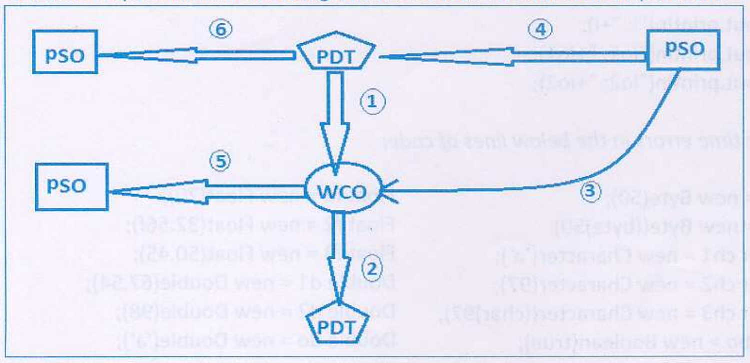
iii

The classes which are used to represent primitive values as object are called wrapper classes. In *java.lang* package we have 8 wrapper classes one per each primitive type to represent them as object. Among them 6 wrapper classes represent number type values these wrapper classes are called number wrapper classes. These 6 number wrapper classes are subclasses of a class ***Number*** which is an abstract class. This class has 6 xxxValue() method to read primitive value from wrapper class object. Except *byteValue()* and *shortValue()* remaining all 4 methods are abstract methods so it is abstract class. These 6 methods are implemented in all 6 number wrapper classes by returning the current object's internal primitive value.

Below diagram shows primitive types and their wrapper classes


Primitive types
Wrapper classes
byte
Byte
short
Short
int
Integer
long
Long
float
Float
double
Double
char
Character
boolean
Boolean
void
Void
Number
public byte byteValue(){}
public short shortValue(){}
public abstract int intValue();
public abstract long longValue();
public abstract float floatValue();
public abstract double doubleValue();
public char charValue();
public boolean booleanValue();

Need of Wrapper classes
Basically wrapper classes are used in project to perform conversion operations. We have 6 conversion operations. Below diagram shows these 6 conversion operations



Wrapper classes constructors and methods
To perform above 6 conversions every wrapper class has the required number of constructors and methods. Check API documentation for more details.

Below diagram show all constructors and methods list

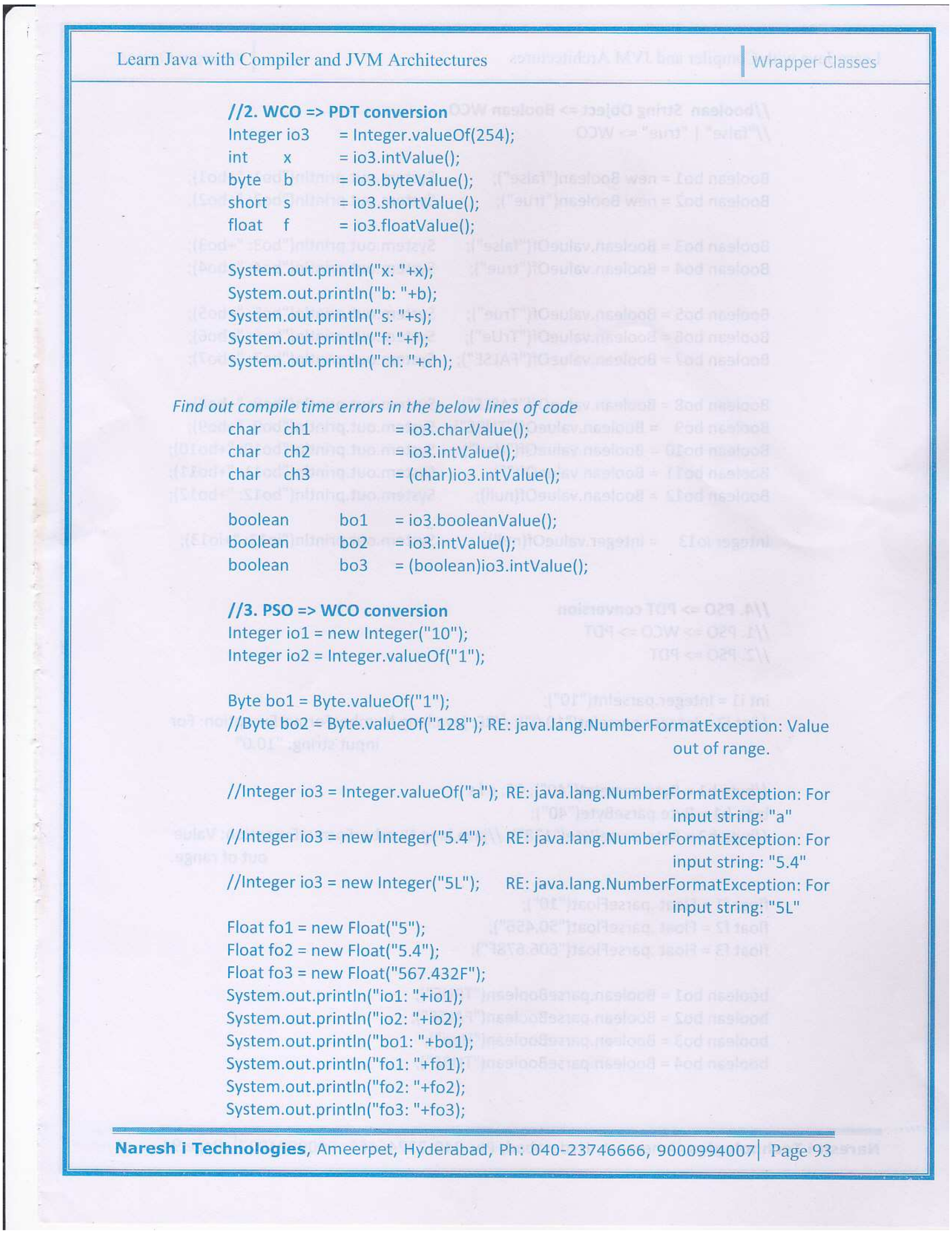
	Constructor(PDT) valueOf(PDT) (static method)	Constructor(S) valueOf(S) (static method)	xxxValue()	parseXxx(S) (static method)	toString(PDT) (static method)	toString()
Byte	✓	✓	✓ 6	✓	✓	✓
Short	✓	✓	✓ 6	✓	✓	✓
Integer	✓	✓	✓ 6	✓	✓	✓
Long	✓	✓	✓ 6	✓	✓	✓
Float	✓ + Float(double)	✓	✓ 6	✓	✓	✓
Double	✓	✓	✓ 6	✓	✓	✓
Character	✓	✗	✓ 1	✗	✓	✓
Boolean	✓	✓	✓ 1	✓	✓	✓

Below program shows performing all above 6 conversions

```
class WrapperClassesDemo {

        public static void main(String[] args) {

                //1. PDT => WCO conversion
                int i = 10;
                Integer io1 = new Integer(i);
                Integer io2 = Integer.valueOf(i);

                System.out.println("i: "+i);
                System.out.println("io1: "+io1);
                System.out.println("io2: "+io2);
```

Find out compile time errors in the below lines of code;

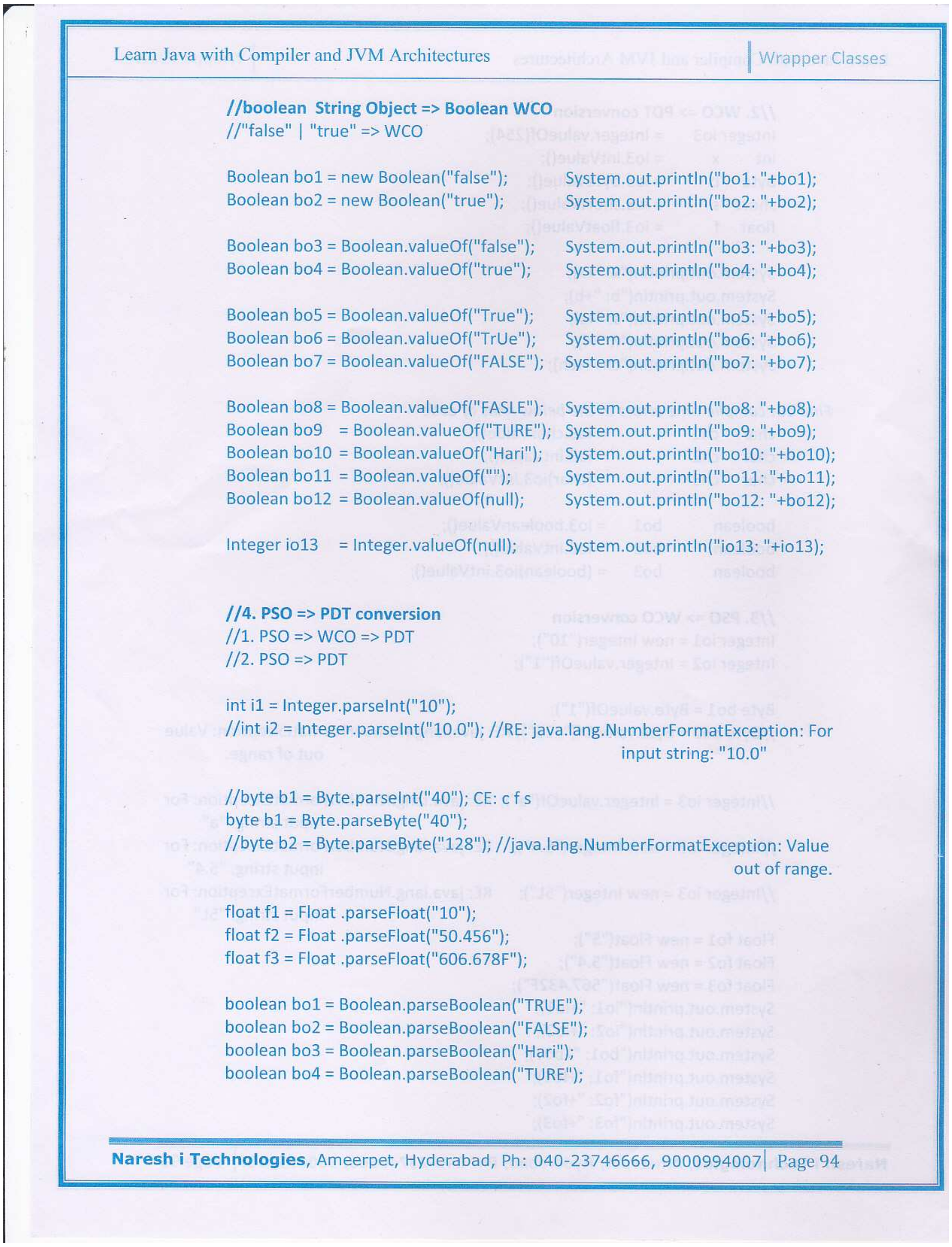
```
Byte b1 = new Byte(50);                      Float f1 = new Float(70);
Byte b2 = new Byte((byte)50);                Float f2 = new Float(32.56f);
Character ch1 = new Character('a');          Float f3 = new Float(50.45);
Character ch2 = new Character(97);           Double d1 = new Double(67.54);
Character ch3 = new Character((char)97);     Double d2 = new Double(98);
Boolean bo = new Boolean(true);              Double do = new Double('a');
```



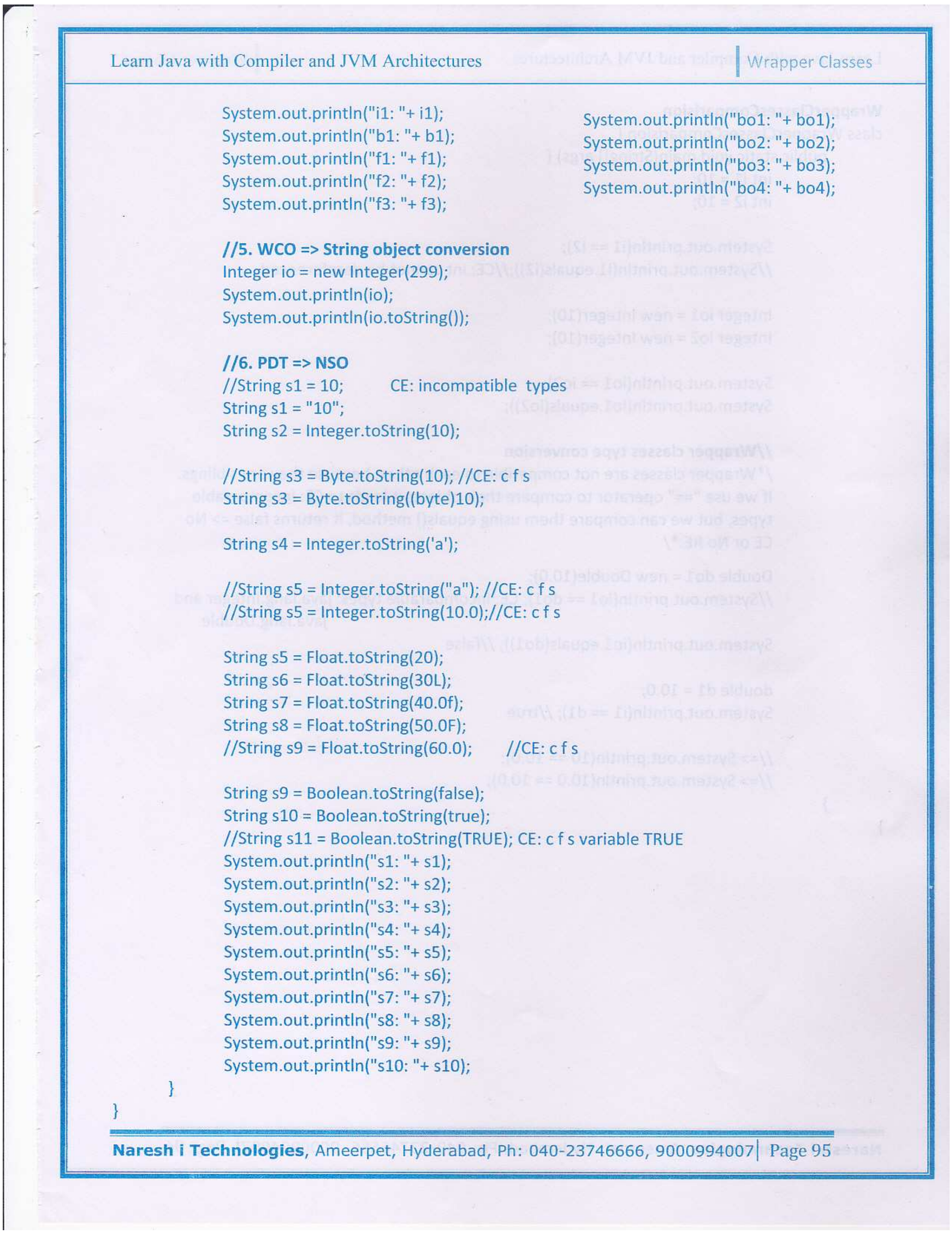
```
//2. WCO => PDT conversion
Integer io3    = Integer.valueOf(254);
int     x      = io3.intValue();
byte    b      = io3.byteValue();
short   s      = io3.shortValue();
float   f      = io3.floatValue();

System.out.println("x: "+x);
System.out.println("b: "+b);
System.out.println("s: "+s);
System.out.println("f: "+f);
System.out.println("ch: "+ch);
```

Find out compile time errors in the below lines of code

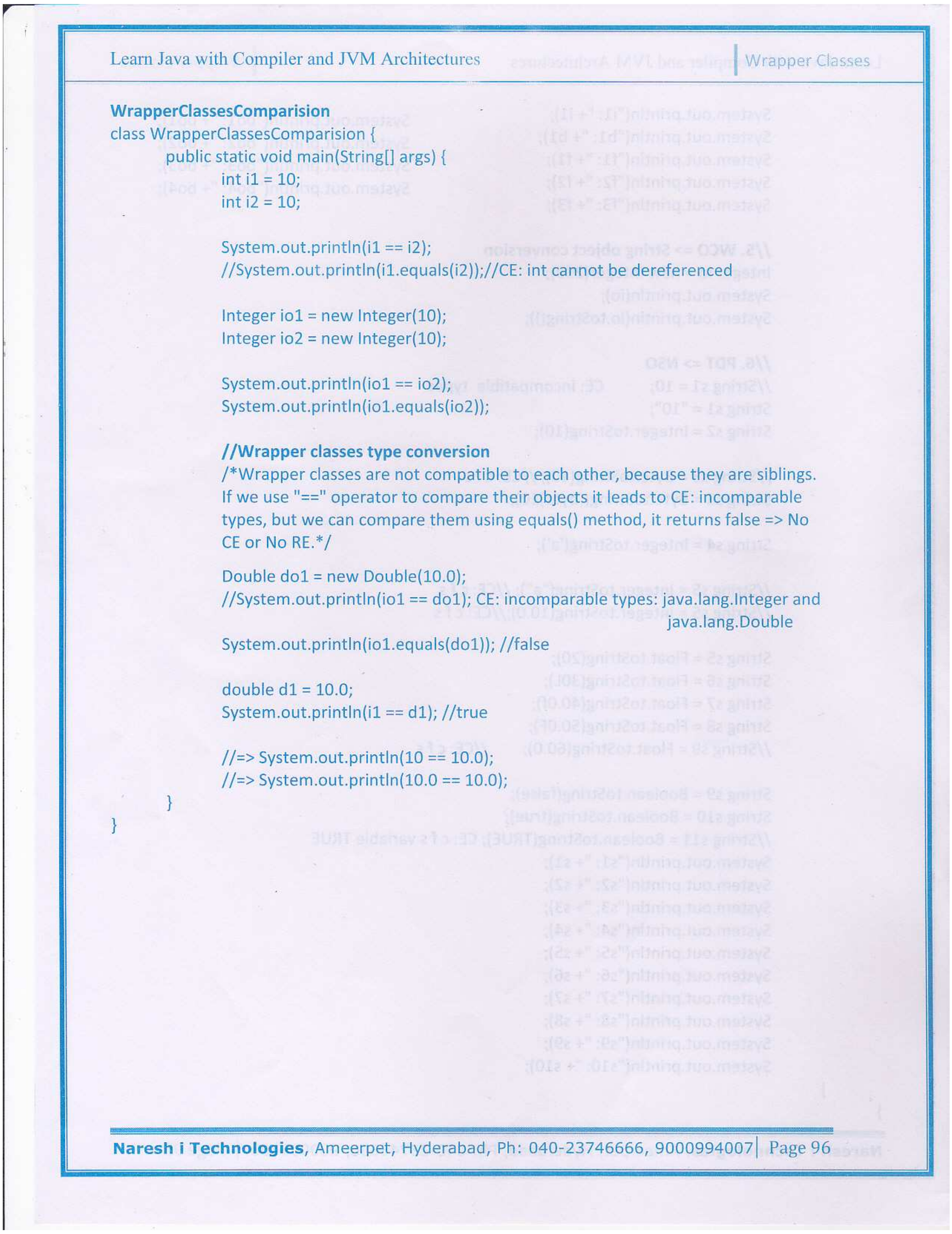
```
char    ch1          = io3.charValue();
char    ch2          = io3.intValue();
char    ch3          = (char)io3.intValue();

boolean         bo1     = io3.booleanValue();
boolean         bo2     = io3.intValue();
boolean         bo3     = (boolean)io3.intValue();

//3. PSO => WCO conversion
Integer io1 = new Integer("10");
Integer io2 = Integer.valueOf("1");

Byte bo1 = Byte.valueOf("1");
//Byte bo2 = Byte.valueOf("128"); RE: java.lang.NumberFormatException: Value
                                                            out of range.

//Integer io3 = Integer.valueOf("a");  RE: java.lang.NumberFormatException: For
                                                      input string: "a"
//Integer io3 = new Integer("5.4");   RE: java.lang.NumberFormatException: For
                                                      input string: "5.4"
//Integer io3 = new Integer("5L");    RE: java.lang.NumberFormatException: For
                                                      input string: "5L"
Float fo1 = new Float("5");
Float fo2 = new Float("5.4");
Float fo3 = new Float("567.432F");
System.out.println("io1: "+io1);
System.out.println("io2: "+io2);
System.out.println("bo1: "+bo1);
System.out.println("fo1: "+fo1);
System.out.println("fo2: "+fo2);
System.out.println("fo3: "+fo3);
```



```
//boolean  String Object => Boolean WCO
//"false" | "true" => WCO

Boolean bo1 = new Boolean("false");     System.out.println("bo1: "+bo1);
Boolean bo2 = new Boolean("true");      System.out.println("bo2: "+bo2);

Boolean bo3 = Boolean.valueOf("false");   System.out.println("bo3: "+bo3);
Boolean bo4 = Boolean.valueOf("true");    System.out.println("bo4: "+bo4);

Boolean bo5 = Boolean.valueOf("True");    System.out.println("bo5: "+bo5);
Boolean bo6 = Boolean.valueOf("TrUe");    System.out.println("bo6: "+bo6);
Boolean bo7 = Boolean.valueOf("FALSE");   System.out.println("bo7: "+bo7);

Boolean bo8 = Boolean.valueOf("FASLE");   System.out.println("bo8: "+bo8);
Boolean bo9   = Boolean.valueOf("TURE");  System.out.println("bo9: "+bo9);
Boolean bo10  = Boolean.valueOf("Hari");  System.out.println("bo10: "+bo10);
Boolean bo11  = Boolean.valueOf("");      System.out.println("bo11: "+bo11);
Boolean bo12  = Boolean.valueOf(null);    System.out.println("bo12: "+bo12);

Integer io13    = Integer.valueOf(null);  System.out.println("io13: "+io13);


//4. PSO => PDT conversion
//1. PSO => WCO => PDT
//2. PSO => PDT

int i1 = Integer.parseInt("10");
//int i2 = Integer.parseInt("10.0"); //RE: java.lang.NumberFormatException: For
                                                      input string: "10.0"

//byte b1 = Byte.parseInt("40"); CE: c f s
byte b1 = Byte.parseByte("40");
//byte b2 = Byte.parseByte("128"); //java.lang.NumberFormatException: Value
                                                              out of range.

float f1 = Float .parseFloat("10");
float f2 = Float .parseFloat("50.456");
float f3 = Float .parseFloat("606.678F");

boolean bo1 = Boolean.parseBoolean("TRUE");
boolean bo2 = Boolean.parseBoolean("FALSE");
boolean bo3 = Boolean.parseBoolean("Hari");
boolean bo4 = Boolean.parseBoolean("TURE");
```

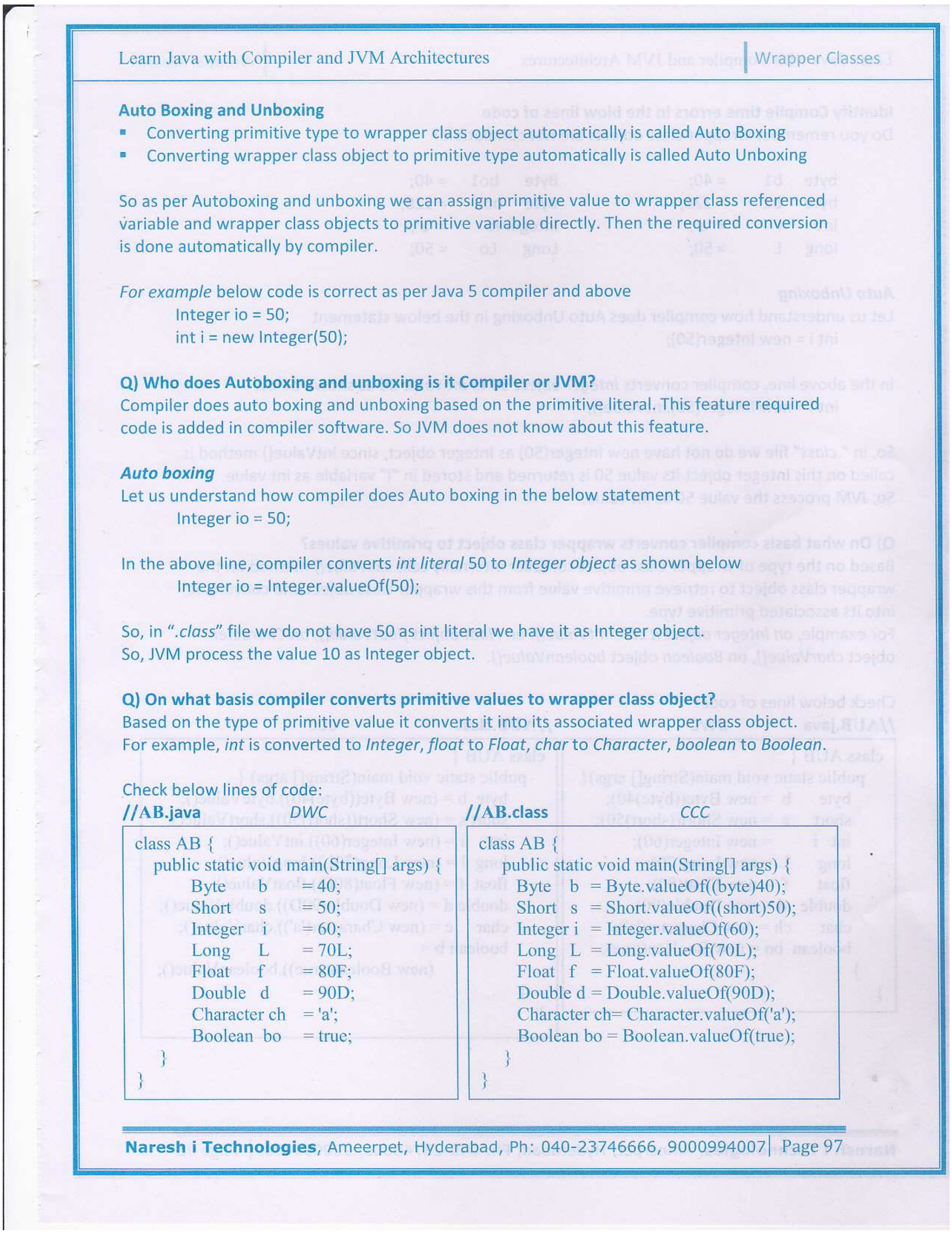


```
        System.out.println("i1: "+ i1);
        System.out.println("b1: "+ b1);
        System.out.println("f1: "+ f1);
        System.out.println("f2: "+ f2);
        System.out.println("f3: "+ f3);

        System.out.println("bo1: "+ bo1);
        System.out.println("bo2: "+ bo2);
        System.out.println("bo3: "+ bo3);
        System.out.println("bo4: "+ bo4);

        //5. WCO => String object conversion
        Integer io = new Integer(299);
        System.out.println(io);
        System.out.println(io.toString());

        //6. PDT => NSO
        //String s1 = 10;        CE: incompatible  types
        String s1 = "10";
        String s2 = Integer.toString(10);

        //String s3 = Byte.toString(10); //CE: c f s
        String s3 = Byte.toString((byte)10);

        String s4 = Integer.toString('a');

        //String s5 = Integer.toString("a"); //CE: c f s
        //String s5 = Integer.toString(10.0);//CE: c f s

        String s5 = Float.toString(20);
        String s6 = Float.toString(30L);
        String s7 = Float.toString(40.0f);
        String s8 = Float.toString(50.0F);
        //String s9 = Float.toString(60.0);       //CE: c f s

        String s9 = Boolean.toString(false);
        String s10 = Boolean.toString(true);
        //String s11 = Boolean.toString(TRUE); CE: c f s variable TRUE
        System.out.println("s1: "+ s1);
        System.out.println("s2: "+ s2);
        System.out.println("s3: "+ s3);
        System.out.println("s4: "+ s4);
        System.out.println("s5: "+ s5);
        System.out.println("s6: "+ s6);
        System.out.println("s7: "+ s7);
        System.out.println("s8: "+ s8);
        System.out.println("s9: "+ s9);
        System.out.println("s10: "+ s10);
    }
}
```



WrapperClassesComparision

```
class WrapperClassesComparision {
        public static void main(String[] args) {
                int i1 = 10;
                int i2 = 10;

                System.out.println(i1 == i2);
                //System.out.println(i1.equals(i2));//CE: int cannot be dereferenced

                Integer io1 = new Integer(10);
                Integer io2 = new Integer(10);

                System.out.println(io1 == io2);
                System.out.println(io1.equals(io2));

                //Wrapper classes type conversion
                /*Wrapper classes are not compatible to each other, because they are siblings.
                If we use "==" operator to compare their objects it leads to CE: incomparable
                types, but we can compare them using equals() method, it returns false => No
                CE or No RE.*/

                Double do1 = new Double(10.0);
                //System.out.println(io1 == do1); CE: incomparable types: java.lang.Integer and
                                                                        java.lang.Double

                System.out.println(io1.equals(do1)); //false

                double d1 = 10.0;
                System.out.println(i1 == d1); //true

                //=> System.out.println(10 == 10.0);
                //=> System.out.println(10.0 == 10.0);
        }
}
```



Auto Boxing and Unboxing

- Converting primitive type to wrapper class object automatically is called Auto Boxing
- Converting wrapper class object to primitive type automatically is called Auto Unboxing

So as per Autoboxing and unboxing we can assign primitive value to wrapper class referenced variable and wrapper class objects to primitive variable directly. Then the required conversion is done automatically by compiler.

For example below code is correct as per Java 5 compiler and above

```
        Integer io = 50;
        int i = new Integer(50);
```

Q) Who does Autoboxing and unboxing is it Compiler or JVM?
Compiler does auto boxing and unboxing based on the primitive literal. This feature required code is added in compiler software. So JVM does not know about this feature.

Auto boxing
Let us understand how compiler does Auto boxing in the below statement

```
        Integer io = 50;
```

In the above line, compiler converts *int literal* 50 to *Integer object* as shown below

```
        Integer io = Integer.valueOf(50);
```

So, in "*.class*" file we do not have 50 as int literal we have it as Integer object.
So, JVM process the value 10 as Integer object.

Q) On what basis compiler converts primitive values to wrapper class object?
Based on the type of primitive value it converts it into its associated wrapper class object.
For example, *int* is converted to *Integer*, *float* to *Float*, *char* to *Character*, *boolean* to *Boolean*.

Check below lines of code:

//AB.java *DWC*

```
class AB {
   public static void main(String[] args) {
         Byte      b      = 40;
         Short     s      = 50;
         Integer   i      = 60;
         Long      L      = 70L;
         Float     f      = 80F;
         Double    d      = 90D;
         Character ch     = 'a';
         Boolean   bo     = true;
   }
}
```

//AB.class *CCC*

```
class AB {
   public static void main(String[] args) {
         Byte    b  = Byte.valueOf((byte)40);
         Short   s  = Short.valueOf((short)50);
         Integer i  = Integer.valueOf(60);
         Long    L  = Long.valueOf(70L);
         Float   f  = Float.valueOf(80F);
         Double d = Double.valueOf(90D);
         Character ch= Character.valueOf('a');
         Boolean bo = Boolean.valueOf(true);
   }
}
```

Identify Compile time errors in the blow lines of code

Do you remember wrapper class objects are not compatible?

```
byte   b1    = 40;          Byte    bo1   = 40;
byte   b2    = 128;         Byte    bo2   = 128;
Int    i     = 'a';         Integer io    = 'a';
long   L     = 50;          Long    Lo    = 50;
```

Auto Unboxing

Let us understand how compiler does Auto Unboxing in the below statement

```
int i = new Integer(50);
```

In the above line, compiler converts *Integer object* 50 to *int value 50* as shown below

```
int i = new Integer(50).intValue();
```

So, in "*.class*" file we do not have new Integer(50) as Integer object, since intValue() method is called on this Integer object its value 50 is returned and stored in "i" variable as int value.
So, JVM process the value 50 as int value.

Q) On what basis compiler converts wrapper class object to primitive values?

Based on the type of wrapper class object compiler internally calls *xxxValue()* method on the wrapper class object to retrieve primitive value from this wrapper class object and converts it into its associated primitive type.
For example, *on Integer object it calls intValue()*, on *Float* object *floatValue()*, on *Character* object *charValue()*, on *Boolean* object *booleanValue()*.

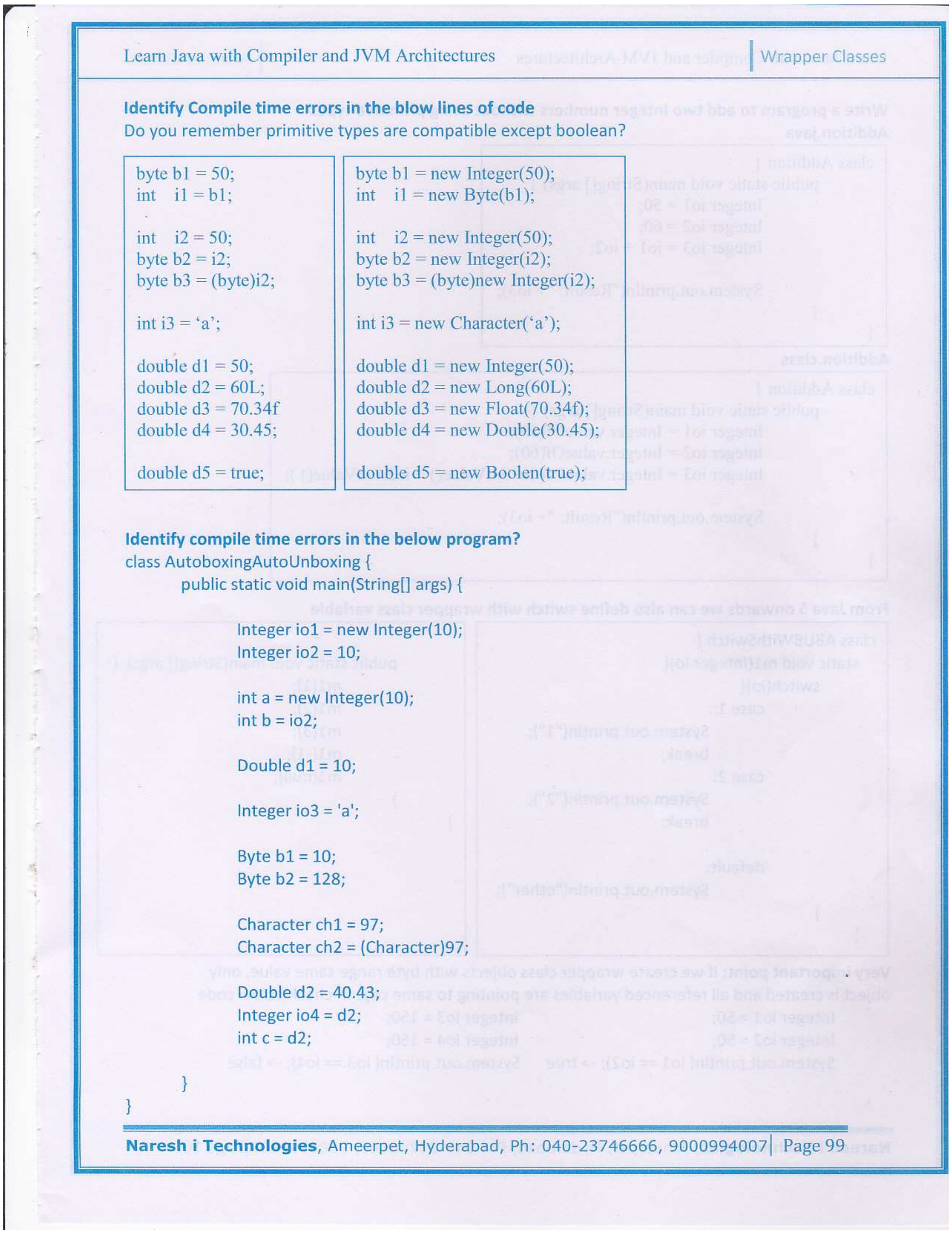
Check below lines of code:

//AUB.java *DWC*

```
class AUB {
  public static void main(String[] args){
    byte    b  = new Byte((byte)40);
    short   s  = new Short((short)50);
    int  i     = new Integer(60);
    long    L = new Long(70);
    float   f = new Float(80);
    double  d = new Double(90);
    char    ch = new Character('a');
    boolean  bo = new Boolean(true);
  }
}
```

//AUB.class *CCC*

```
class AUB {
  public static void main(String[] args) {
    byte  b = (new Byte((byte)40)).byteValue();
    short s = (new Short((short)50)).shortValue();
    int    i = (new Integer(60)).intValue();
    long   l = (new Long(70L)).longValue();
    float  f = (new Float(80F)).floatValue();
    double d = (new Double(90D)).doubleValue();
    char     c = (new Character('a')).charValue();
    boolean b =
                (new Boolean(true)).booleanValue();
  }
}
```



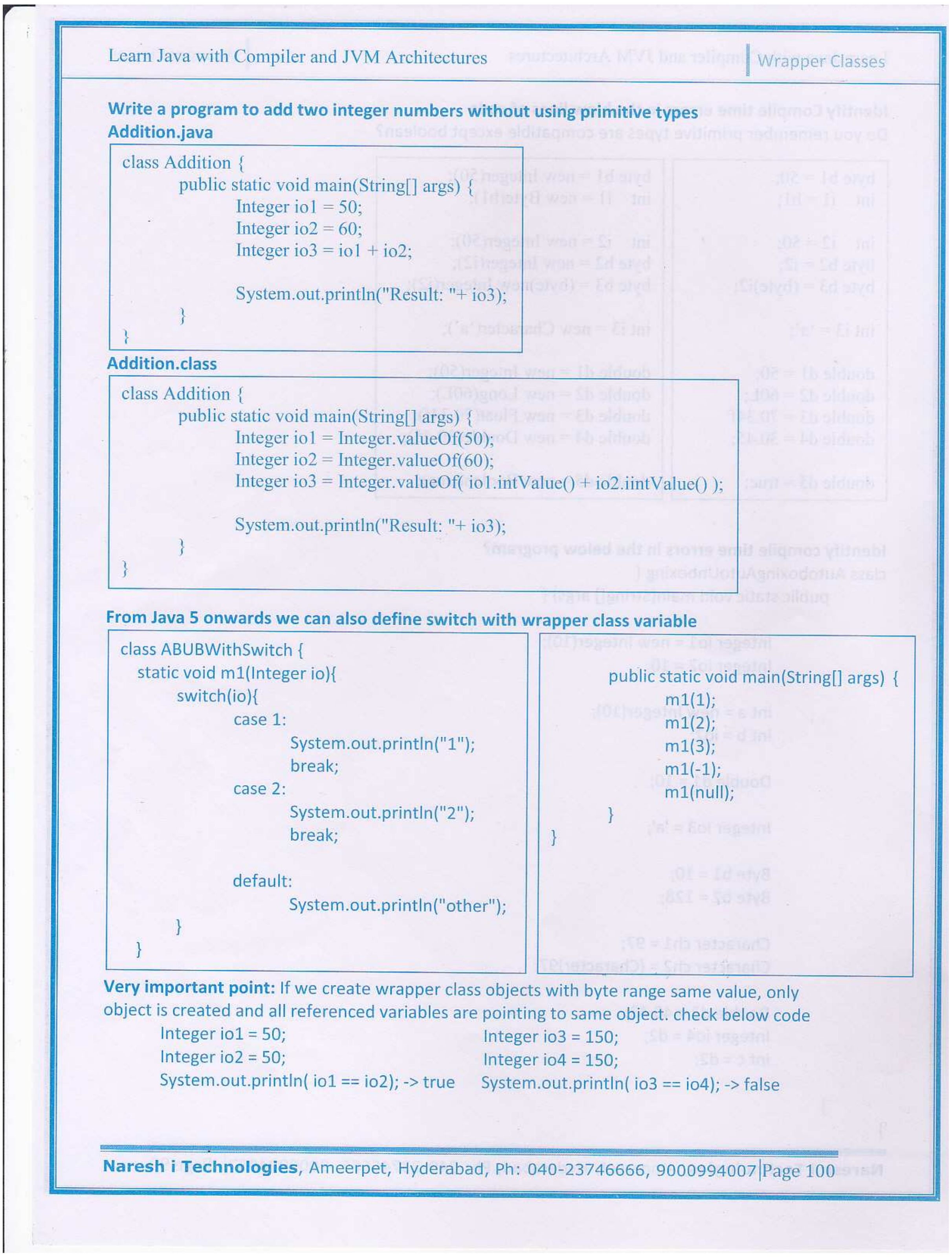
Identify Compile time errors in the blow lines of code
Do you remember primitive types are compatible except boolean?

```
byte b1 = 50;
int   i1 = b1;

int   i2 = 50;
byte b2 = i2;
byte b3 = (byte)i2;

int i3 = 'a';

double d1 = 50;
double d2 = 60L;
double d3 = 70.34f
double d4 = 30.45;

double d5 = true;
```

```
byte b1 = new Integer(50);
int   i1 = new Byte(b1);

int   i2 = new Integer(50);
byte b2 = new Integer(i2);
byte b3 = (byte)new Integer(i2);

int i3 = new Character('a');

double d1 = new Integer(50);
double d2 = new Long(60L);
double d3 = new Float(70.34f);
double d4 = new Double(30.45);

double d5 = new Boolen(true);
```

Identify compile time errors in the below program?

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

                Integer io1 = new Integer(10);
                Integer io2 = 10;

                int a = new Integer(10);
                int b = io2;

                Double d1 = 10;

                Integer io3 = 'a';

                Byte b1 = 10;
                Byte b2 = 128;

                Character ch1 = 97;
                Character ch2 = (Character)97;

                Double d2 = 40.43;
                Integer io4 = d2;
                int c = d2;

        }
}
```



Write a program to add two integer numbers without using primitive types

Addition.java

```
class Addition {
        public static void main(String[] args) {
                Integer io1 = 50;
                Integer io2 = 60;
                Integer io3 = io1 + io2;

                System.out.println("Result: "+ io3);
        }
}
```

Addition.class

```
class Addition {
        public static void main(String[] args) {
                Integer io1 = Integer.valueOf(50);
                Integer io2 = Integer.valueOf(60);
                Integer io3 = Integer.valueOf( io1.intValue() + io2.iintValue() );

                System.out.println("Result: "+ io3);
        }
}
```

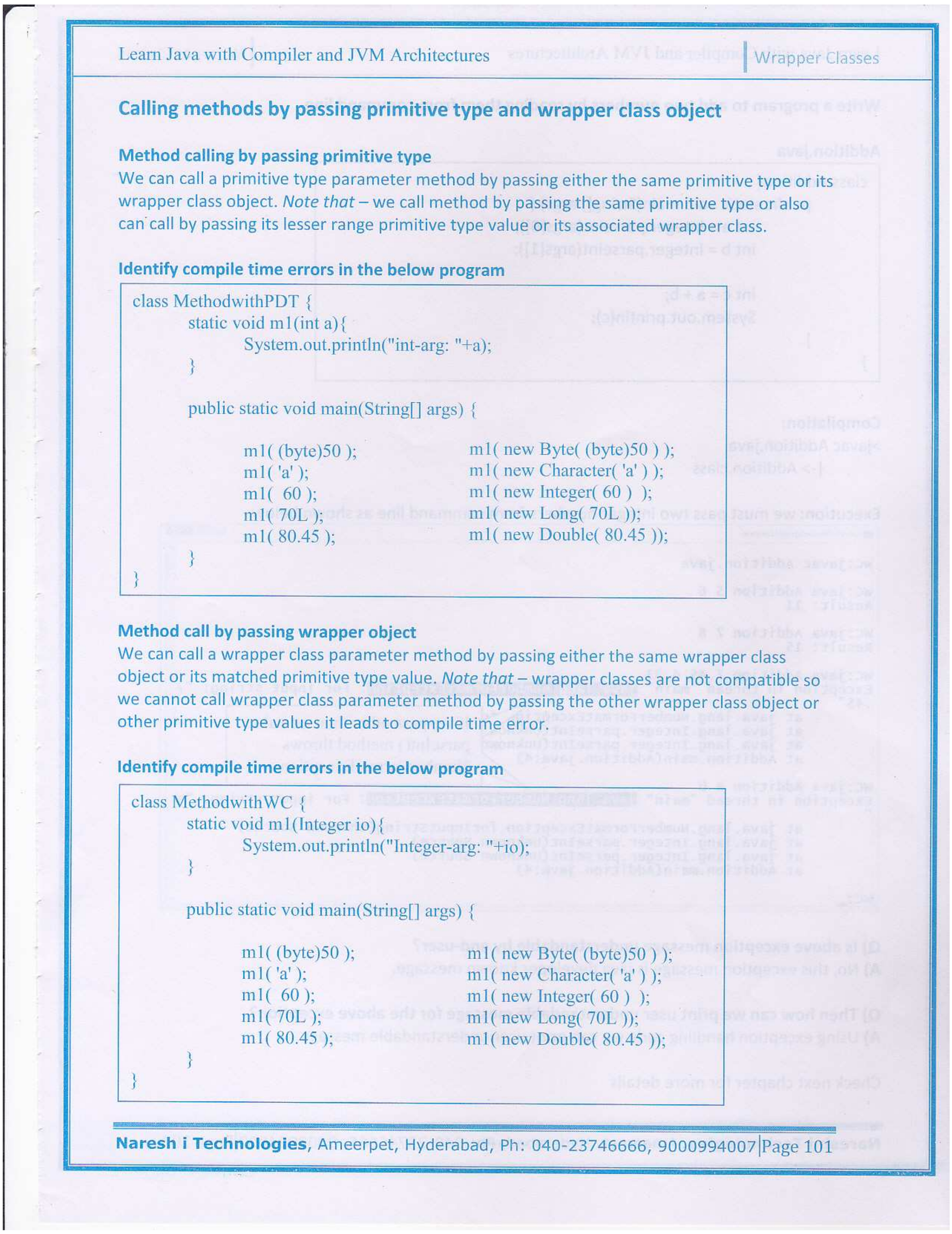
From Java 5 onwards we can also define switch with wrapper class variable

```
class ABUBWithSwitch {
   static void m1(Integer io){
        switch(io){
                case 1:
                        System.out.println("1");
                        break;
                case 2:
                        System.out.println("2");
                        break;

                default:
                        System.out.println("other");
        }
   }
```

```
        public static void main(String[] args) {
                m1(1);
                m1(2);
                m1(3);
                m1(-1);
                m1(null);
        }
}
```

Very important point: If we create wrapper class objects with byte range same value, only object is created and all referenced variables are pointing to same object. check below code

```
Integer io1 = 50;                                   Integer io3 = 150;
Integer io2 = 50;                                   Integer io4 = 150;
System.out.println( io1 == io2); -> true            System.out.println( io3 == io4); -> false
```



Calling methods by passing primitive type and wrapper class object

Method calling by passing primitive type

We can call a primitive type parameter method by passing either the same primitive type or its wrapper class object. *Note that* – we call method by passing the same primitive type or also can call by passing its lesser range primitive type value or its associated wrapper class.

Identify compile time errors in the below program

```
class MethodwithPDT {
        static void m1(int a){
                System.out.println("int-arg: "+a);
        }

        public static void main(String[] args) {

                m1( (byte)50 );           m1( new Byte( (byte)50 ) );
                m1( 'a' );                m1( new Character( 'a' ) );
                m1(  60 );                m1( new Integer( 60 )  );
                m1( 70L );                m1( new Long( 70L ));
                m1( 80.45 );              m1( new Double( 80.45 ));
        }
}
```

Method call by passing wrapper object

We can call a wrapper class parameter method by passing either the same wrapper class object or its matched primitive type value. *Note that* – wrapper classes are not compatible so we cannot call wrapper class parameter method by passing the other wrapper class object or other primitive type values it leads to compile time error.

Identify compile time errors in the below program

```
class MethodwithWC {
        static void m1(Integer io){
                System.out.println("Integer-arg: "+io);
        }

        public static void main(String[] args) {

                m1( (byte)50 );           m1( new Byte( (byte)50 ) );
                m1( 'a' );                m1( new Character( 'a' ) );
                m1(  60 );                m1( new Integer( 60 )  );
                m1( 70L );                m1( new Long( 70L ));
                m1( 80.45 );              m1( new Double( 80.45 ));
        }
}
```

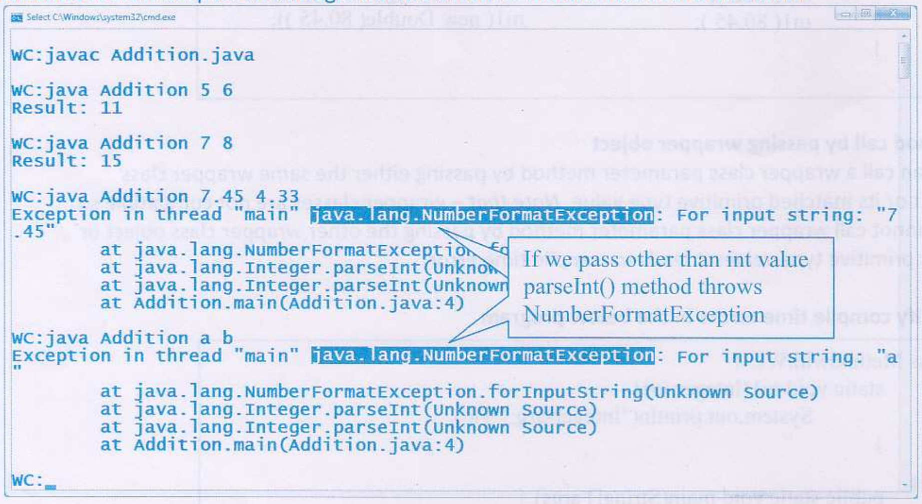
Write a program to add two numbers by reading them from command line

Addition.java

```
class Addition{
        public static void main(String[] args) {
                int a = Integer.parseInt(args[0]);
                int b = Integer.parseInt(args[1]);

                int c = a + b;
                System.out.println(c);
        }
}
```

Compilation:
```
>javac Addition.java
        |-> Addition.class
```

Execution: we must pass two integer numbers from command line as shown below


Select C:\Windows\system32\cmd.exe
If we pass other than int value
parseInt() method throws
NumberFormatException

```
WC:javac Addition.java

WC:java Addition 5 6
Result: 11

WC:java Addition 7 8
Result: 15

WC:java Addition 7.45 4.33
Exception in thread "main" java.lang.NumberFormatException: For input string: "7
.45"
        at java.lang.NumberFormatExceptio
        at java.lang.Integer.parseInt(Unknow
        at java.lang.Integer.parseInt(Unknown
        at Addition.main(Addition.java:4)

WC:java Addition a b
Exception in thread "main" java.lang.NumberFormatException: For input string: "a
"
        at java.lang.NumberFormatException.forInputString(Unknown Source)
        at java.lang.Integer.parseInt(Unknown Source)
        at java.lang.Integer.parseInt(Unknown Source)
        at Addition.main(Addition.java:4)

WC:_
```

Q) Is above exception message understandable by end-user?
A) No, this exception message is java developer known message.

Q) Then how can we print user understandable message for the above exception?
A) Using exception handling code we can print user understandable message.

Check next chapter for more details