**java.lang package – Part-11**

* **Overloading with respect to Autoboxing, widening & var-arg methods:**

**Case\_01: Autoboxing VS Widening**

class Test{

public static void m1(Integer I){

System.out.println(“Auto Boxing”);

}

public static void m1(long l){

System.out.println(“Widening”);

}

public static void main(String[] args){

int x = 10;

m1(x);

}

}

Output: Widening

Note:

Widening dominates autoboxing.

Remind, always the older concept will get the priority with new one, if there is a situation for both the option.

**Case\_02: Widening VS Var-Arg Method:**

class Test{

public static void m1(int… x){

System.out.println(“var-arg method”);

}

public static void m1(long l){

System.out.println(“Widening”);

}

public static void main(String[] args){

int x = 10;

m1(x);

}

}

Output: Widening

Note: Widening dominates var-arg methods.

**Case\_03: Autoboxing vs var-args method:**

class Test{

public static void m1(int… x){

System.out.println(“var-arg method”);

}

public static void m1(Integer I){

System.out.println(“Autoboxing”);

}

public static void main(String[] args){

int x = 10;

m1(10);

}

}

Output: Autoboxing

Note:

Autoboxing dominates var-args methods.

In general var-arg method will get least priority. That is if no other method matched, then only var-arg method will get the chance.

It is exactly same as default case inside switch.

Note\*\*\*

While resolving overloaded methods compiler will always give the precedence in the following order.

1. Widening
2. Autoboxing
3. Var-arg Methods

**Case\_04:**

class Test{

public static void m1(Long l){

System.out.println(“Long”);

}

public static void main(String[] args){

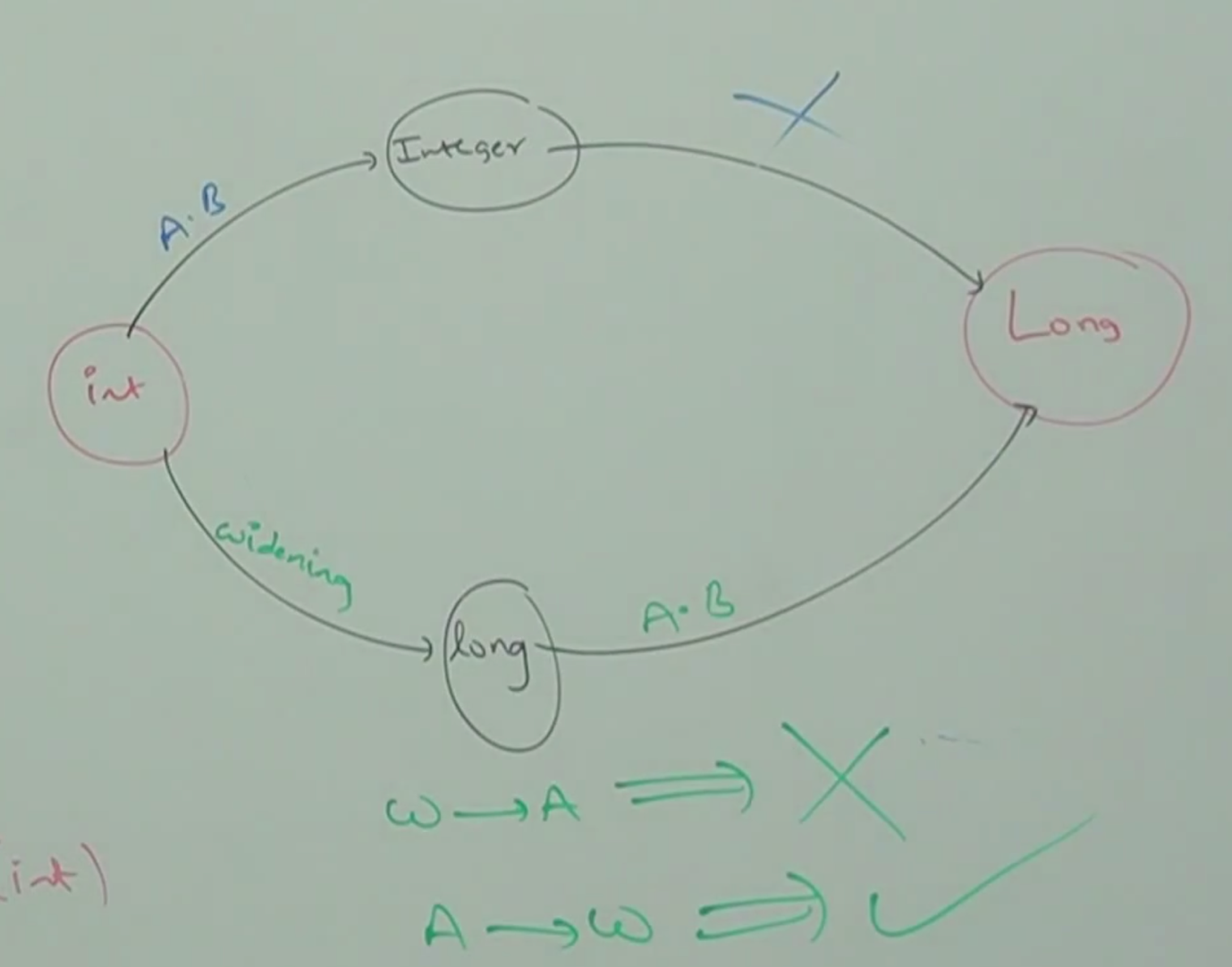
int x = 10;

m1(x);

}

}

Output: CE: m1(java.lang.Long) in Test cannot be applied to (int)



Widening followed by Autoboxing is not allowed in Java.

Whereas Autoboxing followed by widening is allowed.

* **Example\_01:**

Long l = 10; // Invalid, widening and then autoboxing.

CE: incompatible types

found: int

required: java.lang.Long

long l = 10; // Valid, this concept is called widening

* **Example\_02:**

class Test{

public static void m1(Object o){

System.out.println(“Object version”);

}

public static void main(String[] args){

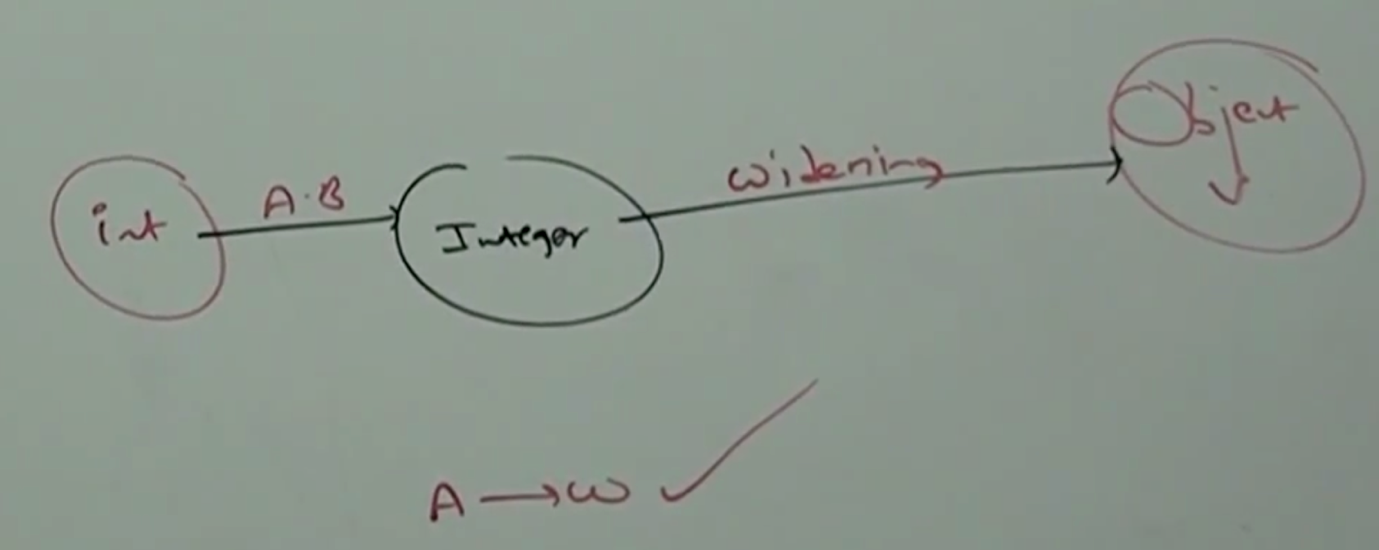
int x = 10;

m1(10);

}

}

Output: Object version



Object o = 10; // Valid

Number n = 10; // Valid

* **Which of the following assignments are legal?**

int I = 10; // Valid

Integer I = 10; // Valid (Autoboxing)

int I = 10L; // Invalid, CE: possible loss of precision, found:long req:int

Long L = 10L; // Valid (Autoboxing)

Long L = 10; //Invalid (CE: Incompatible types found: int required: Long)

long l = 10; // Valid (Widening)

Object o = 10; // valid (Autoboxing followed by Widening)

double d = 10; // Valid (widening)

Doulbe D = 10; // Invalid (CE: Incompatible types found: int req:Double

Number n = 10; // Valid (Autoboxing followed by Widening)