Autoboxing and auto-unboxing

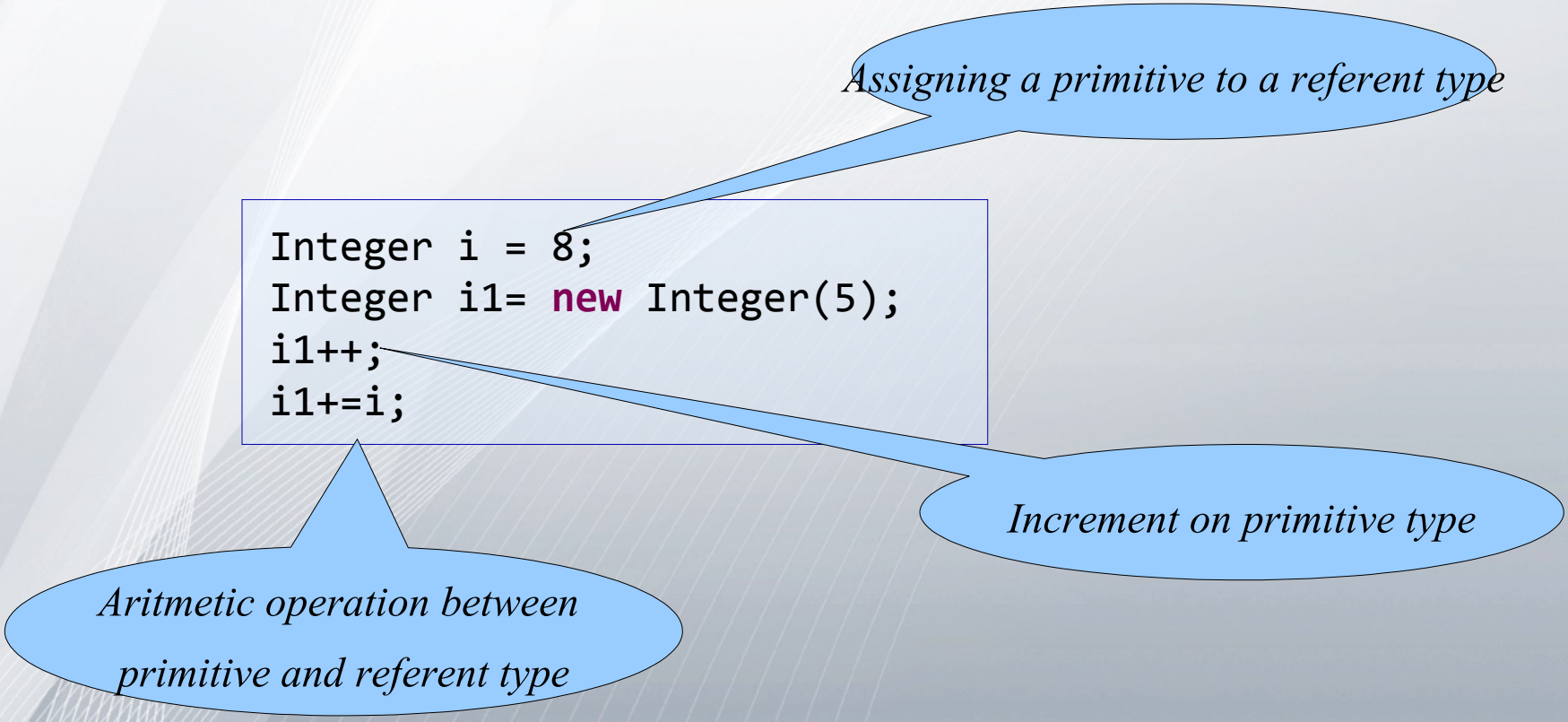
Referent types for primitives

- In the java language the primitive types have their numeric presentation as referent types:
  - Integer for int
  - Double for Double
  - Float for float
  - Byte for byte
  - Long for long
  - Short for short
  - And BigDecimal and BigInteger

Referent types for primitives

- The usage of referent types is necessary for the object presentation of the primitives

BUT



Collections and primitives

```
List list = new ArrayList(10);
for(int i=0;i<10;i++){
    list.add(i);
}

Iterator it = list.iterator();
while(it.hasNext()){
    int i =(Integer)it.next();
}
```

List and all other collections work with objects...but here a primitive is added

A cast to Integer works with int

Autoboxing and Unboxing

The process of automatic transformation of a primitive to an object is called Autoboxing

And the reverse process is Unboxing

Boxing and Unboxing exist in Java since Java 1.5

The two processes make it possible to use numeric types disregarding the type – primitive or referent

Autoboxing and unboxing

```
List list = new ArrayList(10);
for(int i=0;i<10;i++){
    list.add(i);
}

Iterator it = list.iterator();
while(it.hasNext()){
    int i =(Integer)it.next();
}
```

A list of numbers between 1 and 10

```
Integer i = 8;
Integer i1= new Integer(5);
i1++;
i1+=i;
```

i=8

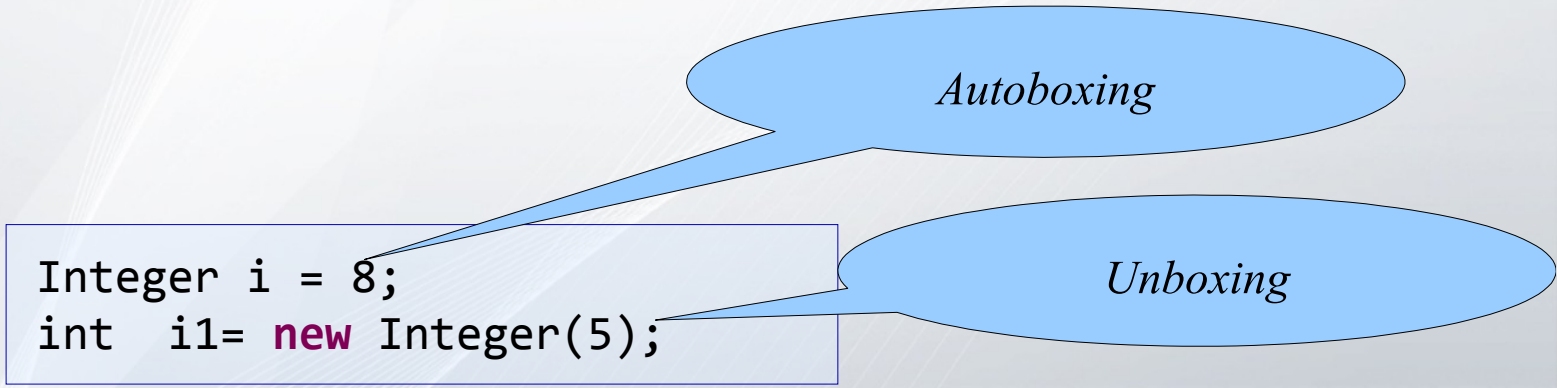
i1=5

i1=6

I1=14

//6+8

Referent types for primitives



More on boxing

- Boxing makes it possible to use referent types as primitives and vice versa
- Warning:
  - Referent types are references – they may throw NullPointerException
  - == operator compares the values for primitives and the reference for referent types

More on boxing

- Boxing and unboxing is not a fast operation
- Don't use it for critical systems and performance-vulnerable systems
- Safes on code
- Useful methods
- When to use it?
  - Mismatch between primitives and referent types