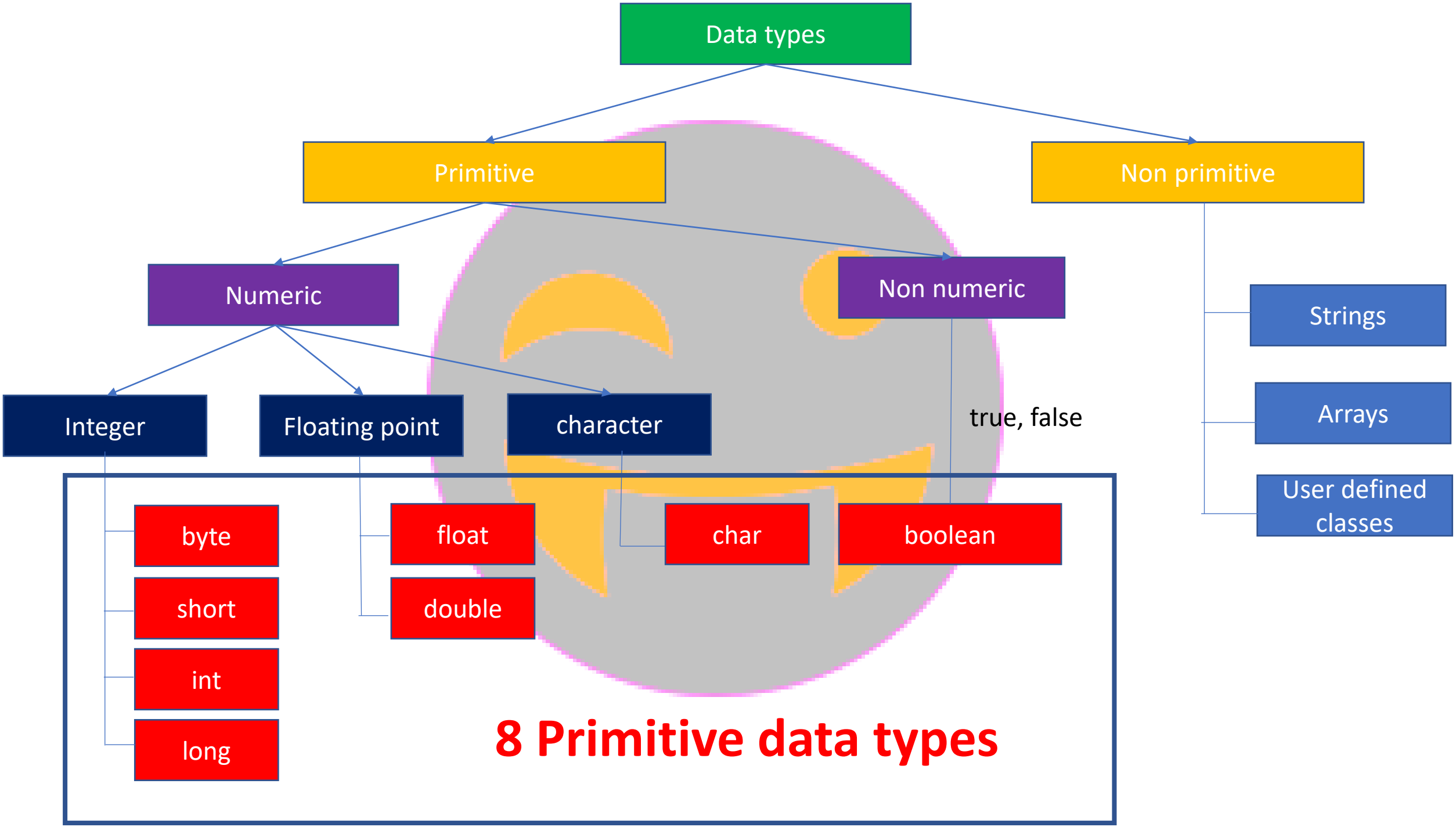
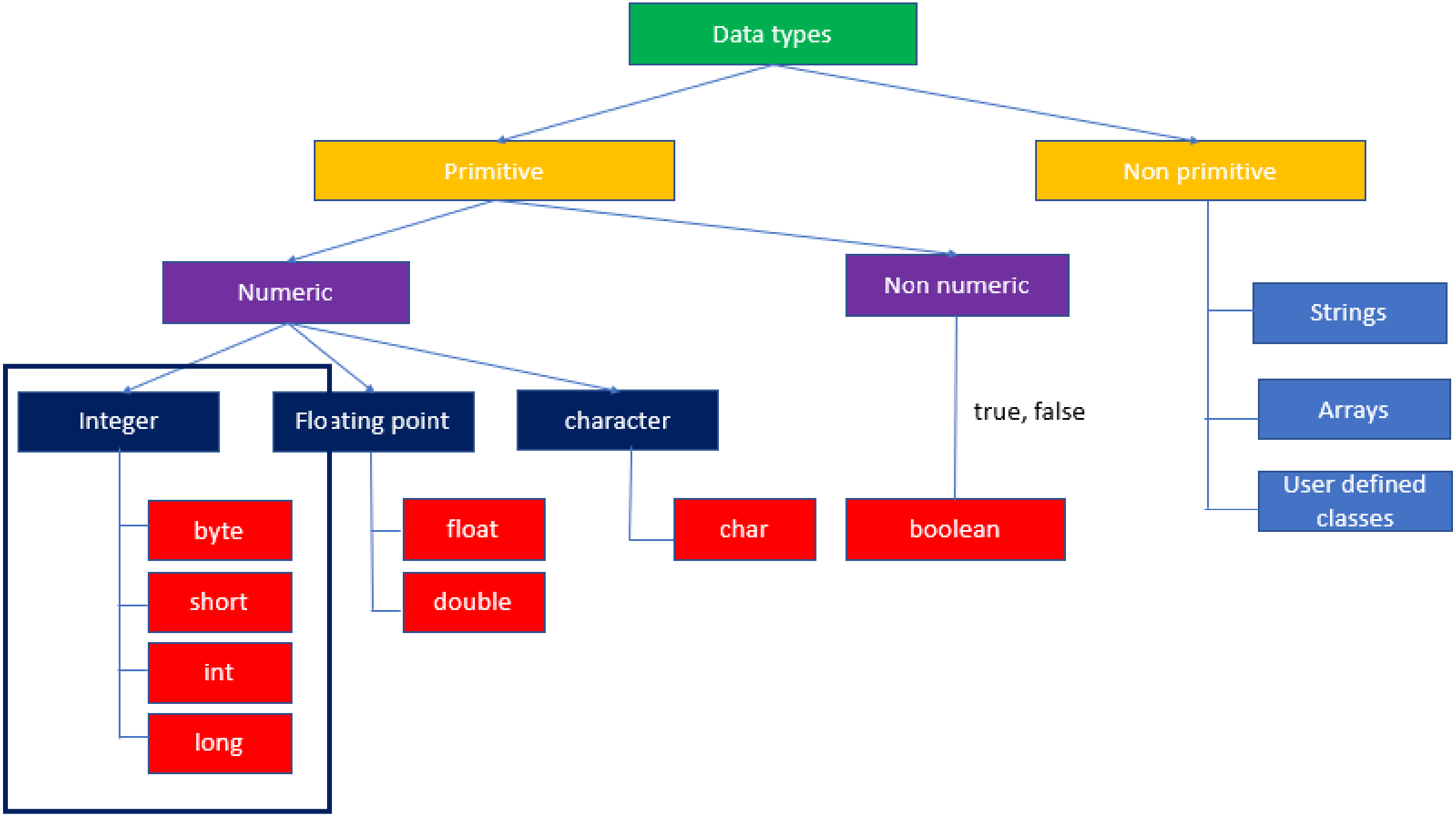
Chapter 20

Data types in Java - Practical - Integers


Data types
Primitive
Non primitive
Numeric
Non numeric
Strings
Arrays
User defined classes
Integer
Floating point
character
true, false
byte
short
int
long
float
double
char
boolean
8 Primitive data types


Data types
Primitive
Non primitive
Numeric
Non numeric
Strings
Arrays
User defined classes
Integer
Floating point
character
true, false
byte
short
int
long
float
double
char
boolean

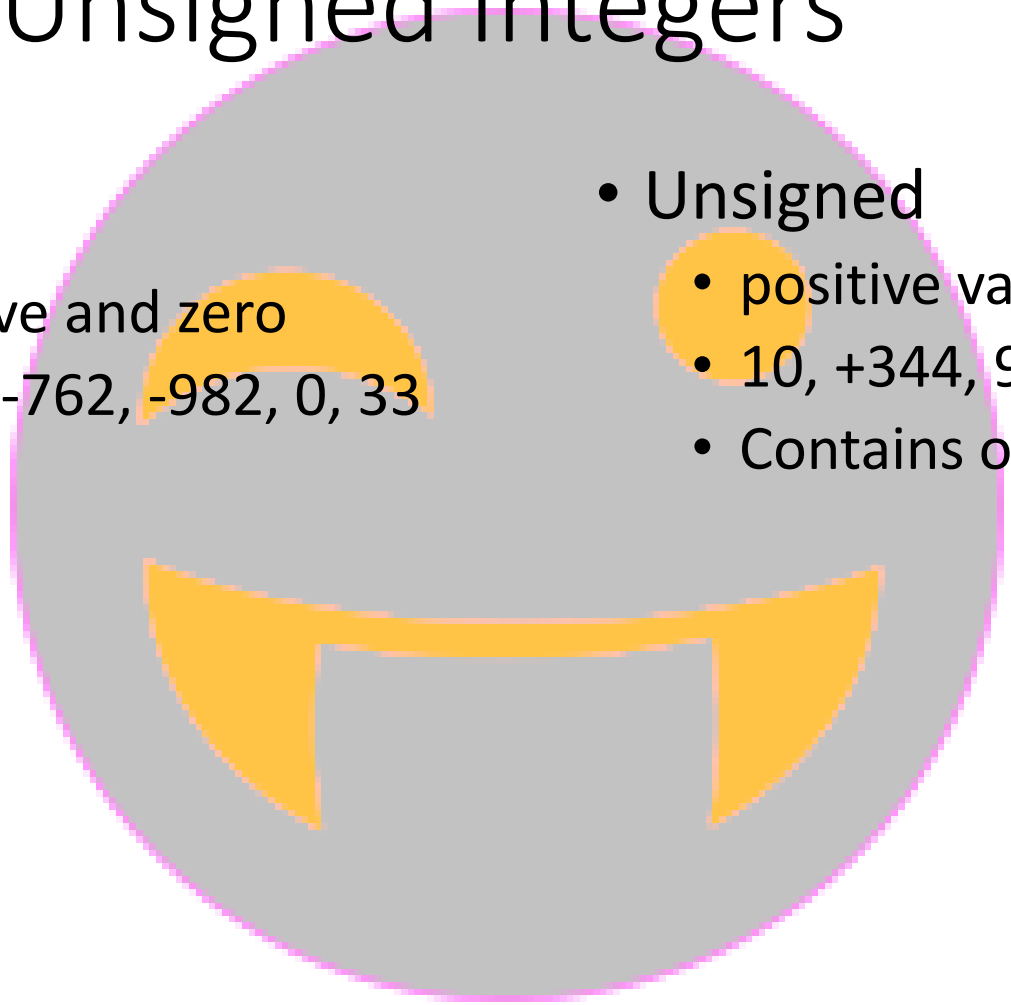
Integer Types(**byte, short, int, long**)

- Used to store **whole numbers** such as **positive(143), negative(-83)** or **zero(0)** without decimals.
- **What data type to be used** will be discussed in sometime
- **NOTE:** Java uses **signed 2's complement to store Integers**

```java
class VariablesDemo{
    int marks = 100;
    public static void main(String[] args){
        System.out.println("Variables Demo");
        int a, b, c, d;
        a = 10;
        b = 20;
        c = -20;
        System.out.println(a);
        System.out.println(b);
        System.out.println(c);
        d = 1000;
        System.out.println(d);
```

Signed and Unsigned Integers

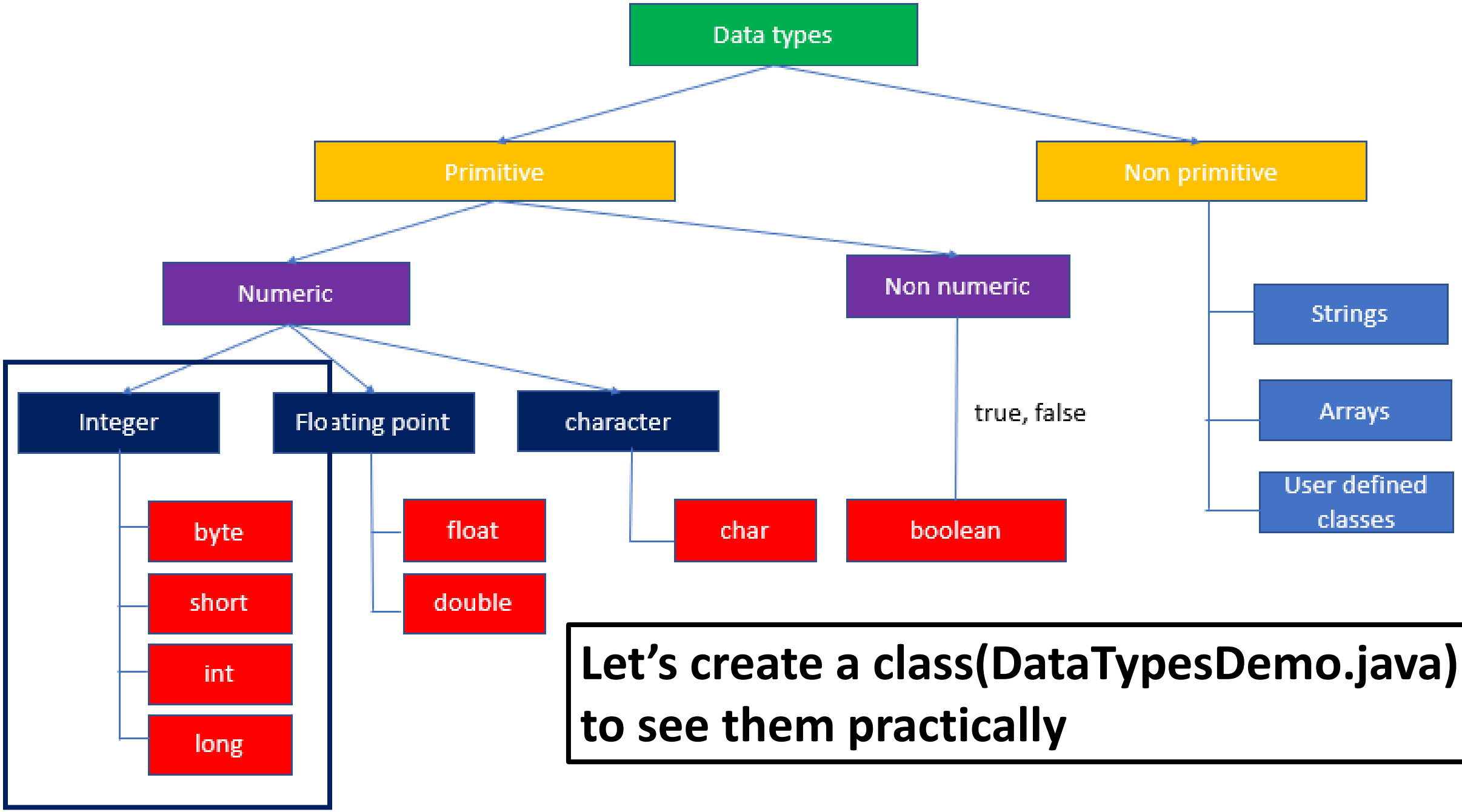
- Signed
  - Positive, negative and zero
  - 78,93, 2976, -9,-762, -982, 0, 33



- Unsigned
  - positive values and zero
  - 10, +344, 93, 0, 4424, 224
  - Contains only non-negative values

Signed and Unsigned Integers

Signed Integers	Unsigned Integers
Positive, negative and zero	0 and positive values
78,93, 2976, -9,-762, -982, 0, 33	10, +344, 93, 0, 4424, 224
	Contains only non-negative values



Let's create a class(DataTypesDemo.java) to see them practically

Program

```
class DataTypesDemo {
    public static void main(String args[]){
        System.out.println("Data Types Demo");
    }
}
```

```
class DataTypesDemo {
    byte basket1;
    public static void main(String args[]){
        System.out.println("Data Types Demo");
        DataTypesDemo demo = new DataTypesDemo();
        System.out.println(demo.basket1);
    }
}
```

1. Create local variable and check if we can print it without assigning a value.
2. We can't do it
3. Let's create an instance variable and print it in the static method
4. Let's access basket1 by creating instance of the class

Program – Byte demo

```
class DataTypesDemo {
    byte basket1;
    public static void main(String args[]){
        System.out.println("Data Types Demo");
        DataTypesDemo demo = new DataTypesDemo();
        System.out.println(demo.basket1);
        //demo.basket1 = 200;//we get error
        System.out.println(demo.basket1);
        demo.basket1 = -128; //allows -128 to 127
        //can I keep a string?
        //demo.basket1 = "suresh";//error
        demo.basket1 = 'a';
        System.out.println(demo.basket1);
        //min and max values
        System.out.println(Byte.MIN_VALUE);
        System.out.println(Byte.MAX_VALUE);
    }
}
```

Create a separate method for the byte demo

```
class DataTypesDemo {
    byte basket1;
    public static void main(String args[]){
        System.out.println("Data Types Demo");
        DataTypesDemo demo = new DataTypesDemo();
        demo.bytesDemo();
    }

    void bytesDemo(){
        System.out.println(basket1);
        //demo.basket1 = 200;//we get error
        System.out.println(basket1);
        basket1 = -128; //allows -128 to 127
        //can I keep a string?
        //demo.basket1 = "suresh";//error
        basket1 = 'a';
        System.out.println(basket1);
        //min and max values
        System.out.println(Byte.MIN_VALUE);
        System.out.println(Byte.MAX_VALUE);
    }
}
```

```java
class DataTypes {
    byte byteBasket;
    short shortBasket;
    public static void main(String args[]){
        System.out.println("Data Types Demo");
        DataTypes d = new DataTypes();
        System.out.println("Default value of byte: "+d.byteBasket);
        d.byteBasket = ' ';
        System.out.println(d.byteBasket);
        System.out.println(Byte.MIN_VALUE);
        System.out.println(Byte.MAX_VALUE);
        System.out.println("Default value of short: "+d.shortBasket);
        d.shortBasket = 2929;
        System.out.println(d.shortBasket);
        System.out.println(Short.MIN_VALUE);
        System.out.println(Short.MAX_VALUE);
    }

    void byteInfo(){

    }
}
```

```java
class DataTypes {
    byte byteBasket;
    short shortBasket;
    public static void main(String args[]){
        System.out.println("Data Types Demo");
        DataTypes d = new DataTypes();
        d.byteInfo();
        d.shortInfo();
    }

    void byteInfo(){
        heading("Byte");
        System.out.println("Default value: "+byteBasket);
        byteBasket = ' ';
        System.out.println(byteBasket);
        System.out.println(Byte.MIN_VALUE);
        System.out.println(Byte.MAX_VALUE);
    }

    void shortInfo(){
        heading("Short");
        System.out.println("Default value: "+shortBasket);
        shortBasket = 2929;
        System.out.println(shortBasket);
        System.out.println(Short.MIN_VALUE);
        System.out.println(Short.MAX_VALUE);
    }

    void heading(String message){
        System.out.println("======"+message+"========");
    }
}
```

byte

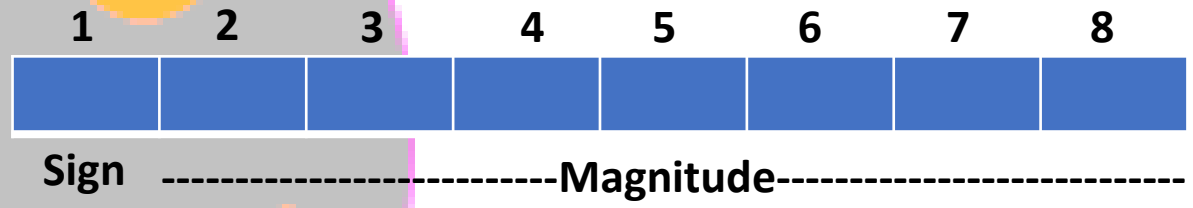
Integers

Size: 1 byte

Range: -128 to 127

Default value: 0

- 1 byte – 8 bits
- 2^8 = 256 numbers
- **8-bit signed two's complement integer**
- We will have positive(+) & negative numbers(-)
- 128 positive and 128 negative
- -128, -127, -126…0, 1, 2, …127
- **-2^7 to 2^7-1**
- So, 1 bit is used for sign(positive, negative)
- **0 – positive**
- **1 – negative**



MIN and MAX values

- **Byte**: **Byte** is a class that **comes with java library** which is called as **boxed primitive or wrapper type**
- Every primitive type(byte, short, int, long, float, double, char, boolean) has an associated wrapper type(boxed primitive)
- **Byte.MIN_VALUE**
- **Byte.MAX_VALUE**

Wrapper types/Boxed primitives

byte	**Byte**	1. java.lang.Byte
short	**Short**	2. java.lang.Short
int	**Integer**	3. java.lang.Integer
long	**Long**	4. java.lang.Long
float	**Float**	5. java.lang.Float
double	**Double**	6. java.lang.Double
char	**Character**	7. java.lang.Character
boolean	**Boolean**	8. java.lang.Boolean

Short demo

```java
void bytesDemo(){
    System.out.println("Default value: "+basket1);
    //demo.basket1 = 200;//we get error
    System.out.println(basket1);
    basket1 = -128; //allows -128 to 127
    //can I keep a string?
    //demo.basket1 = "suresh";//error
    basket1 = 'a';
    System.out.println(basket1);
    //min and max values
    System.out.println("Byte min: "+Byte.MIN_VALUE);
    System.out.println("Byte max: "+Byte.MAX_VALUE);
    System.out.println("==========================");
}

void shortDemo(){
    System.out.println("Default value: "+basket2);
    //basket2 = 32768;//error
    System.out.println(basket2);
    basket2 = 32767;//error(try other values as well)
    System.out.println(basket2);
    basket2 = '*'; //character
    System.out.println(basket2);
    //min and max values
    System.out.println("Short min: "+Short.MIN_VALUE);
    System.out.println("Short max: "+Short.MAX_VALUE);
    System.out.println("==========================");
}
```

```java
class DataTypes {
    byte byteBasket;
    short shortBasket;
    public static void main(String args[]){
        System.out.println("Data Types Demo");
        DataTypes d = new DataTypes();
        d.byteInfo();
        d.shortInfo();
    }

    void byteInfo(){
        heading("Byte");
        System.out.println("Default value: "+byteBasket);
        byteBasket = ' ';
        System.out.println(byteBasket);
        System.out.println(Byte.MIN_VALUE);
        System.out.println(Byte.MAX_VALUE);
    }

    void shortInfo(){
        heading("Short");
        System.out.println("Default value: "+shortBasket);
        shortBasket = 2929;
        System.out.println(shortBasket);
        System.out.println(Short.MIN_VALUE);
        System.out.println(Short.MAX_VALUE);
    }

    void heading(String message){
        System.out.println("======"+message+"========");
    }
}
```

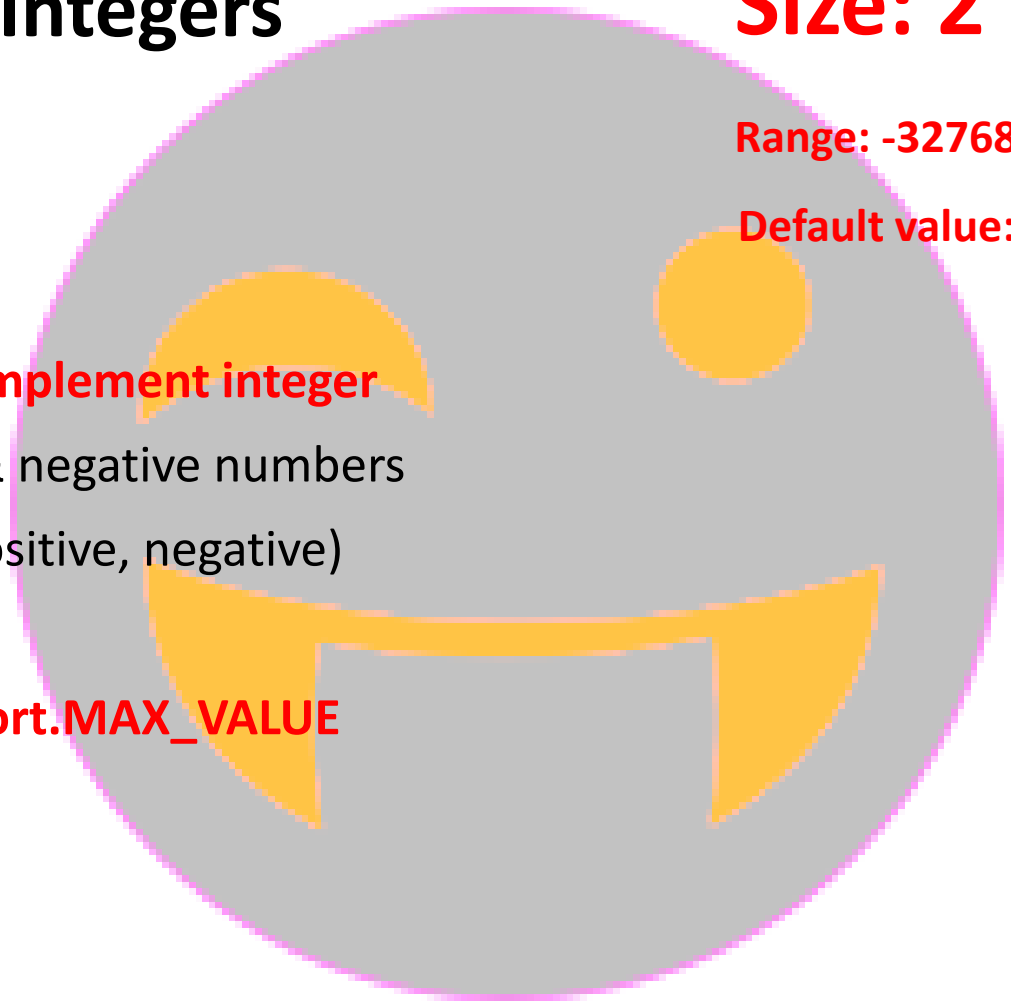
short  **Integers**

Size: 2 bytes

Range: -32768 to 32767

Default value: 0

- 2 bytes – 16 bits
- 2^16 = 65536 numbers
- **16-bit signed two's complement integer**
- We will have positive & negative numbers
- 1 bit is used for sign(positive, negative)
- **-2^15 to 2^15-1**
- **Short.MIN_VALUE, Short.MAX_VALUE**



Integer demo

```
void intDemo(){
    System.out.println("Default value: "+basket3);
    //basket2 = 32768;//error
    System.out.println(basket3);
    basket3 = 2147483648;//error(try other values as well)
    //-error message is quite different(integer number too large)
    //basket3 = 2147483648;//seperate by _ (can't keep at the beginning or ending)
    System.out.println(basket3);
    basket3 = 'A'; //character
    System.out.println(basket3);
    //min and max values
    System.out.println("Short min: "+Integer.MIN_VALUE);
    System.out.println("Short max: "+Integer.MAX_VALUE);
    System.out.println("============================");
}
```

int

Integers

- 4 bytes – 32 bits
- 2^32 = 4,29,49,67,296 numbers
- **32-bit signed two's complement integer**
- We will have positive & negative numbers
- 1 bit is used for sign(positive, negative)
- **-2^31 to 2^31-1**

Size: 4 bytes

Range: -2,14,74,83,648 to 2,14,74,83,647

Default value: 0

NOTE: In Java SE 8 and later, you can use the int data type to represent unsigned 32 bit integer, which has a minimum value of 0 and maximum value of 2^32-1. Use Integer class to use int data type as unsigned integer.

But we don't use it 99.9% so don't worry much.

```
@Native public static final int    MIN_VALUE = 0x80000000;
```

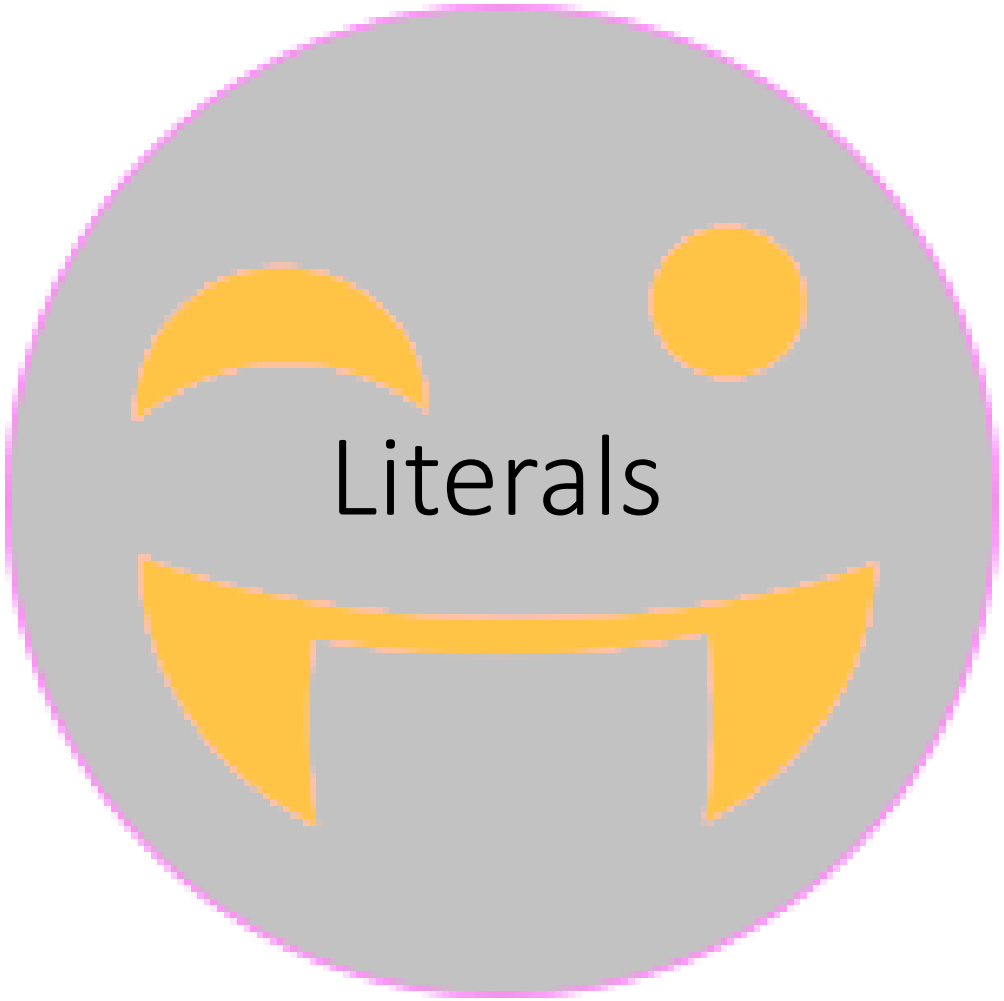
Why is the error different this time?

```
error: incompatible types: possible lossy conversion from int to byte
```

```
error: incompatible types: possible lossy conversion from int to short
```

```
error: integer number too large
```

- You should understand about the **literals**



Literals

Literals

- Any constant value that can be assigned to the variable is called a **literal**
- Integer literal
- Floating-point literal
- Char literal
- String literal
- Boolean literal

```
class CashProgram{
    public static void main(String[] args){
        int suitcase1;
        int suitcase2 = 2000;
        int suitcase3 = 3000;
        suitcase1 = 1000;
```

variablename = literal/expression

suitcase1 = 1000 (literal)

suitcase1 = suitcase2+suitcase3

We c
assig

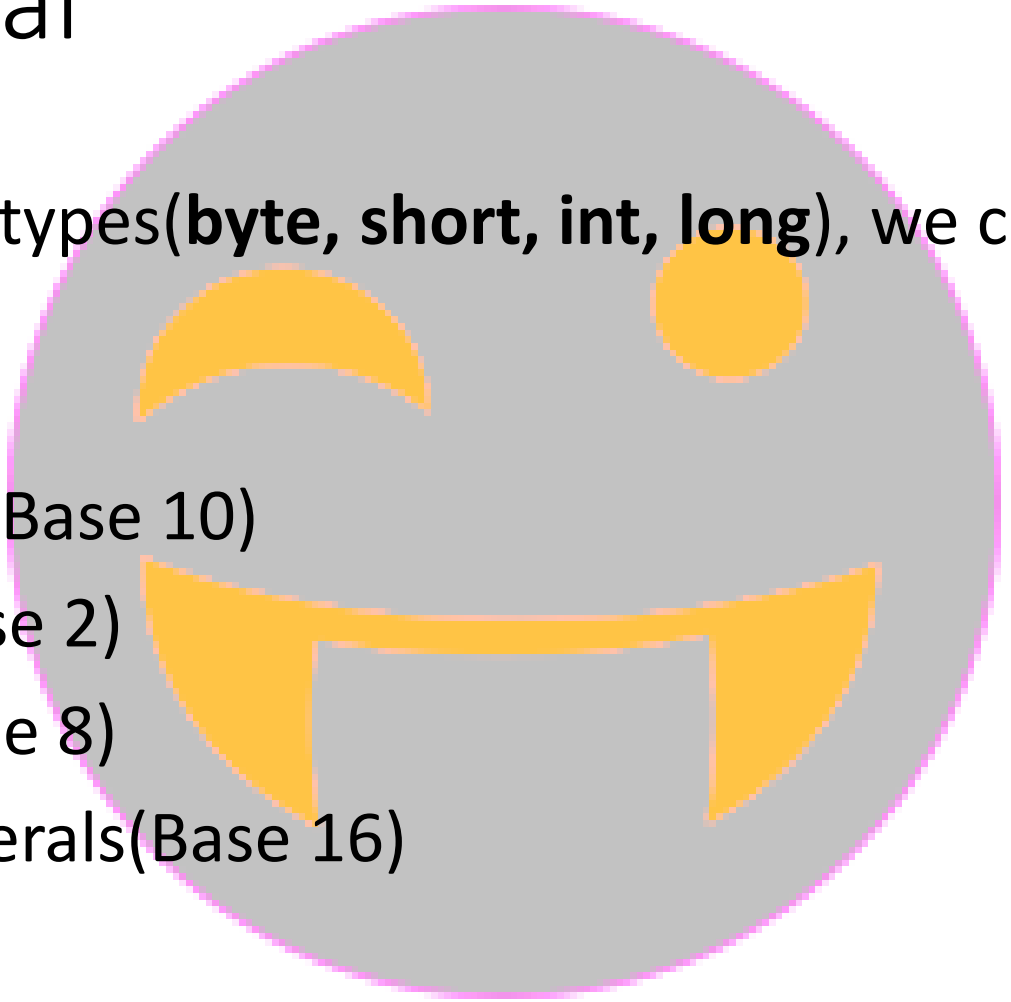
Literals demo

```
void literalsDemo(){
    //int marks = 99;//Decimal value
    //int marks = 0b1100011//binary value;
    //int marks = 0143; //octal
    //int marks =0x63; //hexa decimal
    System.out.println(marks);
}
```

Integer literal

- For Integer data types(**byte, short, int, long**), we can specify literals in 4 ways

- Decimal literals (Base 10)
- Binary literal(Base 2)
- Octal literals(Base 8)
- Hexa-decimal literals(Base 16)



Decimal literals (Base 10)

- Allowed digits are 0-9 (decimal values – ex: 10, 3342, 8373, 99983)

```
void intDemo(){
    //Decimal
    int decVar = 99;
    System.out.println(decVar);

}
```

Binary literals(Base 2)

- Allowed digits are 0 and 1
- Why is there no change?
- Should be prefixed with 0b or 0B

```java
void intDemo(){
    //Decimal
    int decVar = 99;
    System.out.println(decVar);
    //Binary
    int binVar = 10101010;
    System.out.println(binVar);
}
```

```java
void intDemo(){
    //Decimal
    int decVar = 99;
    System.out.println(decVar);
    //Binary
    int binVar = 0b10101010;
    System.out.println(binVar);
}
```

Octal literals (Base 8)

- Allowed digits are 0-7
- Should be prefixed with 0(zero)

```
void intDemo(){
    //Decimal
    int decVar = 99;
    System.out.println(decVar);
    //Binary
    int binVar = 0b10101010;
    System.out.println(binVar);
    //Octal
    int octVar = 342;
    System.out.println(octVar);
}
```

```
void intDemo(){
    //Decimal
    int decVar = 99;
    System.out.println(decVar);
    //Binary
    int binVar = 0b10101010;
    System.out.println(binVar);
    //Octal
    int octVar = 0342;
    System.out.println(octVar);
}
```

Hexa-decimal literal (Base 16)

- Allowed digits are 0-9 and characters are a-f (a, b, c, d, e, f)
- Both upper and lower case characters are allowed(A,B,C,D,E,F)
- Should be prefixed with 0X or 0x

```
void intDemo(){
    //Decimal
    int decVar = 99;
    System.out.println(decVar);
    //Binary
    int binVar = 0b10101010;
    System.out.println(binVar);
    //Octal
    int octVar = 0342;
    System.out.println(octVar);
    //Hexa-decimal
    int hexVar = 0x80000000;
    System.out.println(hexVar);
}
```

int

Integers

- 4 bytes – 32 bits
- 2^32 = 4,29,49,67,296 numbers
- **32-bit signed two's complement integer**
- We will have positive & negative numbers
- 1 bit is used for sign(positive, negative)
- **-2^31 to 2^31-1**

Size: 4 bytes

Range: -2,14,74,83,648 to 2,14,74,83,647

Default value: 0

NOTE: In Java SE 8 and later, you can use the int data type to represent unsigned 32 bit integer, which has a minimum value of 0 and maximum value of 2^32-1. Use Integer class to use int data type as unsigned integer.

But we don't use it 99.9% so don't worry much.

```
@Native public static final int   MIN_VALUE = 0x80000000;
```

```
@Native public static final int   MAX_VALUE = 0x7fffffff;
```

Why is the error different this time?

```
error: incompatible types: possible lossy conversion from int to byte
```

```
error: incompatible types: possible lossy conversion from int to short
```

```
error: integer number too large
```

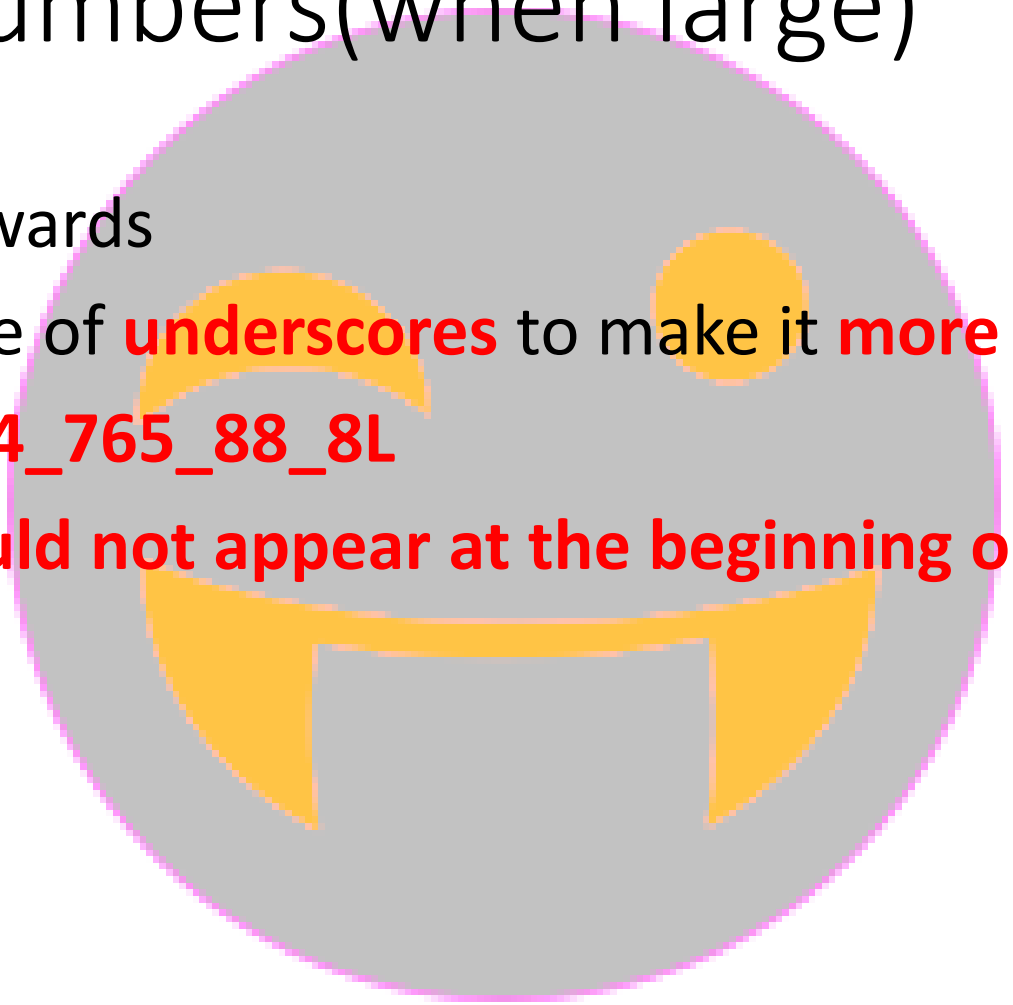
- You should understand more about the **literals**

Note

- By default, **every literal is of type int**
- Integer literals can **either be represented as an int or long**
- **We can specify explicitly as long type by suffixing with l or L**
- That's why we get error message "**incompatible types: possible lossy conversion from int to byte/int to short**" but it will not happen when you assign out of range values to an **int variable**
- There is **no way to specify byte or short literals explicitly** but indirectly we can specify
- Whenever we are assigning integral literal to the byte variable and if the value is within the range of byte then the compiler treats it automatically as byte literals

Readable numbers(when large)

- From **Java 7** on wards
- We can make use of **underscores** to make it **more readable**
- long phone = **234_765_88_8L**
- Underscore **should not appear at the beginning or end**



Invalid (long literal can’t be assigned to byte, short and int)

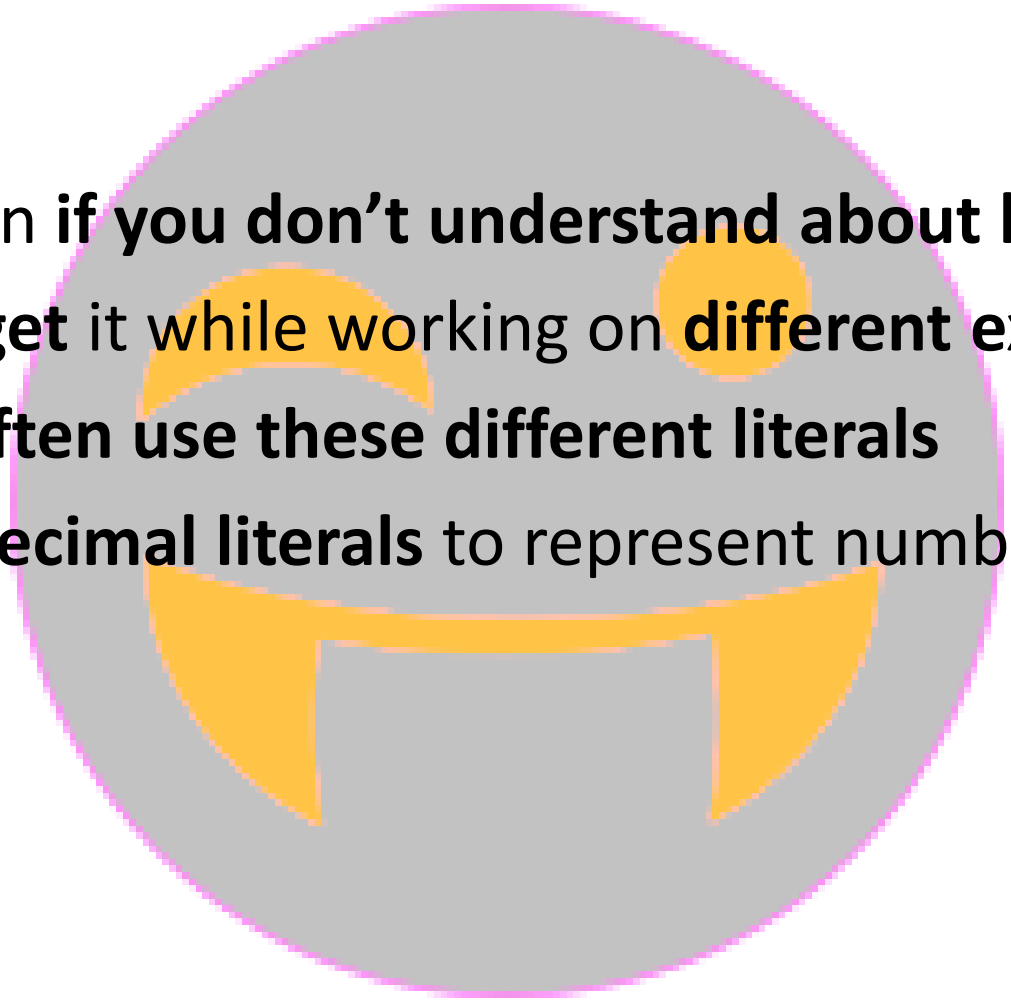
- byte age = 16L (invalid)
- int id = 12L (invalid)
- short rank = 234L (invalid)
- Byte, short and int can only be assigned with int literals
- A long literal can be assigned to long or other data types which are of larger range which we will see later
- Integer literals: int literal and long literal
- int literal can be assigned to all the four data types but the long literal can only be assigned to long

Review

Type	Bit depth	Range	Default values
byte	1 byte(8 bits)	-2^7 to 2^7-1	0
short	2 bytes(16 bits)	-2^15 to 2^15-1	0
int	4 bytes(32 bits)	-2^31 to 2^31-1	0

Note

- **Don’t worry** even **if you don’t understand about literals.**
- Slowly **you will get** it while working on **different examples**
- And, we **don’t often use these different literals**
- Mostly we use **decimal literals** to represent numbers



long  **Integers**

Size: 8 bytes

Range: -92,23,37,20,36,85,47,75,808 to 92,23,37,20,36,85,47,75,807

Default value: 0

- 8 bytes – 64 bits
- 2^64 = 1,84,46,74,40,73,70,95,51,616 numbers
- **64-bit signed two's complement integer**
- We will have positive & negative numbers
- 1 bit is used for sign(positive, negative)
- **-2^63 to 2^63-1**

NOTE: In Java SE 8 and later, you can use the long data type to represent unsigned 64 bit long, which has a minimum value of 0 and maximum value of 2^64-1. Use Long class to use long data type as unsigned long.

But we don't use it 99.9% so don't worry much.

More about literals

- https://docs.oracle.com/javase/specs/jls/se8/html/jls-3.html#jls-3.10
- We can assign an integer literal to byte, short, int and long if it is within the range
- If we assign more than the maximum integer number in the long data type we get error
- Long literal can be denoted by suffixing it with l or L
- We can’t assign a long literal to byte, short, or int even if it is within the range

Integer types

Type	Bit depth	Range	Default values
byte	1 byte(8 bits)	-2^7 to 2^7-1	0
short	2 bytes(16 bits)	-2^15 to 2^15-1	0
int	4 bytes(32 bits)	-2^31 to 2^31-1	0
long	8 bytes(64 bits)	-2^63 to 2^63-1	0

When to use what while programming?

byte(1 byte)

-128 to 127

short(2 bytes)

-32768 to 32767

long (8 bytes)

-92,23,37,20,36,85,47,75,808 to 92,23,37,20,36,85,47,75,807

int (4 bytes)

-2,14,74,83,648 to 2,14,74,83,647

Which basket would Suresh pick?

- Very small, small, medium, or large
- It depends on your requirement

Quantity

10 cloths

20 cloths

40 cloths

80 cloths

Guess which data type is best to pick as per the requirements given

Number of students in a college?

byte

-128 to 127

short

-32768 to 32767

long

-92,23,37,20,36,85,47,75,808 to 92,23,37,20,36,85,47,75,807

int

-2,14,74,83,648 to 2,14,74,83,647

short noOfStudents;
int noOfStudents;

Day of birth

byte

-128 to 127

short

-32768 to 32767

long

-92,23,37,20,36,85,47,75,808 to 92,23,37,20,36,85,47,75,807

int

-2,14,74,83,648 to 2,14,74,83,647

byte dayOfBirth;

Marks of a student

byte

-128 to 127

short

-32768 to 32767

long

-92,23,37,20,36,85,47,75,808 to 92,23,37,20,36,85,47,75,807

int

-2,14,74,83,648 to 2,14,74,83,647

short marks;

Eamcet rank

byte

-128 to 127

short

-32768 to 32767

long

-92,23,37,20,36,85,47,75,808 to 92,23,37,20,36,85,47,75,807

int

-2,14,74,83,648 to 2,14,74,83,647

int emacetRank

Number of views for a video

byte

-128 to 127

short

-32768 to 32767

long

-92,23,37,20,36,85,47,75,808 to 92,23,37,20,36,85,47,75,807

int

-2,14,74,83,648 to 2,14,74,83,647

What if you want to store longer than the range of long data type?

BigInteger

https://docs.oracle.com/javase/6/docs/api/java/math/BigInteger.html

BigInteger

- It is used when integers involved are larger than the limit of long data type
- When the value is too big for int and long data type to handle
- Math operations: add, subtract, multiply, divide

```
import java.math.BigInteger;

void longDemo(){
    //longVar = 9223372036854775807 2L;
    BigInteger bI = new BigInteger("9223372036854775807 2");
    System.out.println(bI);
}
```

Important point to be noted

- Java uses signed 2’s complement to store Integer(byte, short, int, long) numbers
- https://medium.com/@jeanvillete/java-makes-use-of-the-twos-complement-for-representing-signed-numbers-31e421725c04
- Java uses IEEE 7 scheme to store fractional numbers
- Java uses UTF-16 scheme to store characters

Notes - Integer Types(**byte, short, int, long**)

- Used **to store whole numbers such as positive(143), negative(-83) or zero(0) without decimals**.
- What data type to be picked is **dependent on our requirement**
- **[IMPORTANT]** Though there are four data types to store integers, we **will mostly use int data type**
- **Use byte and short if the memory is major concern(to save memory)**
- **NOTE:** Java uses **signed 2's complement to store Integers**

Signed two’s complement

Note:

1. Just for your reference.
2. Nothing will happen even if you don’t understand it
3. If you don’t have much time, then you can happily skip it

What next?

Signed two’s complement – how computer stores integers internally

చిన్న బ్రేక్ చిటికలో వచ్చేస్తా