ABESIT
COLLEGE OF ENGINEERING

CRC-Training

Java™

Core Java

Data types

Variable (Cup) can be of two type

- **Primitive Type**
- **Reference Type**

Primitive Type

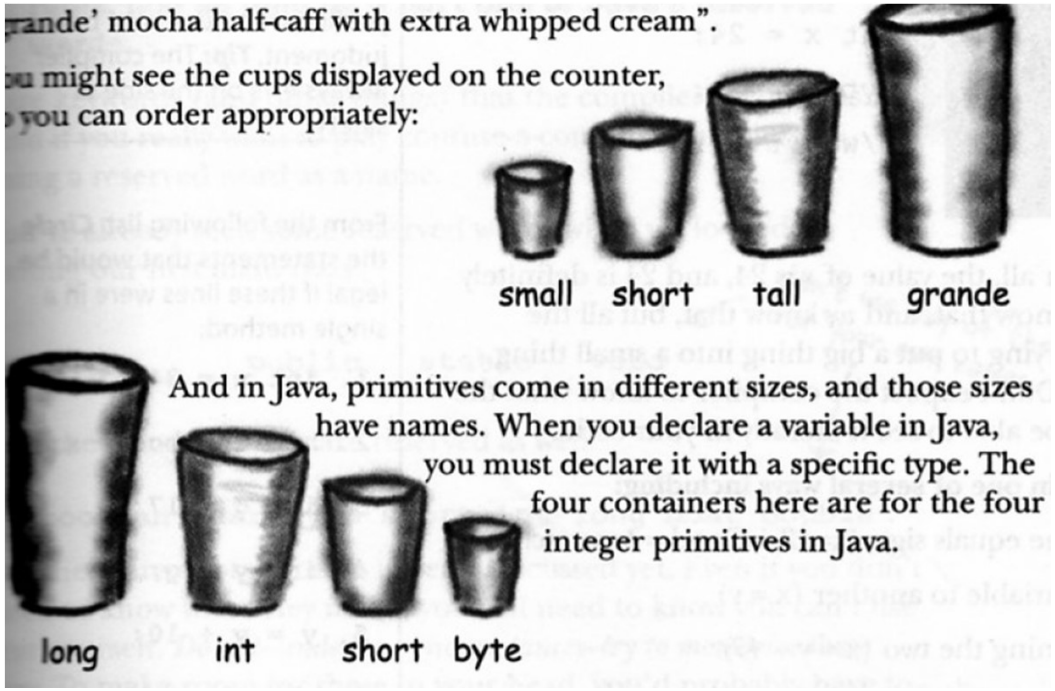
- Primitives are like the cups at the Coffee-House.
- They come in different sizes and each has name like small , big, medium.

Primitive Type Example

```
byte x = 10;        // Take 1 byte in memory
short y = 500;      // Take 2 bytes in memory
int a = 40000;      // Take 4 bytes in memory
```

Data types

❖ Primitive types



- There are **8** primitive types in java.
- Below table summarizes the categories of the primitive types:

Category	Primitive types
integers	byte, short, int, long
floating point numbers	float, double
Characters	char
Booleans	boolean

Data types

- Below table summarizes the corresponding wrapper classes, size, range, and default values:

Primitive type	Wrapper type	Size (in bytes)	Range	Default values
byte	java.lang.Byte	1	-128 to +127	0
short	java.lang.Short	2	-32,768 to + 32,767	0
int	java.lang.Integer	4	-2^{31} to $+2^{31} - 1$	0
long	java.lang.Long	8	-2^{63} to $+2^{63} - 1$	0
float	java.lang.Float	4	-3.4e38 to 3.4e38	0.0F
double	java.lang.Double	8	-1.7e308 to 1.7e308	0.0
char	java.lang.Character	2	0 to 65,535	Blank Space(‘ ‘)
boolean	java.lang.Boolean	N/A	N/A	False

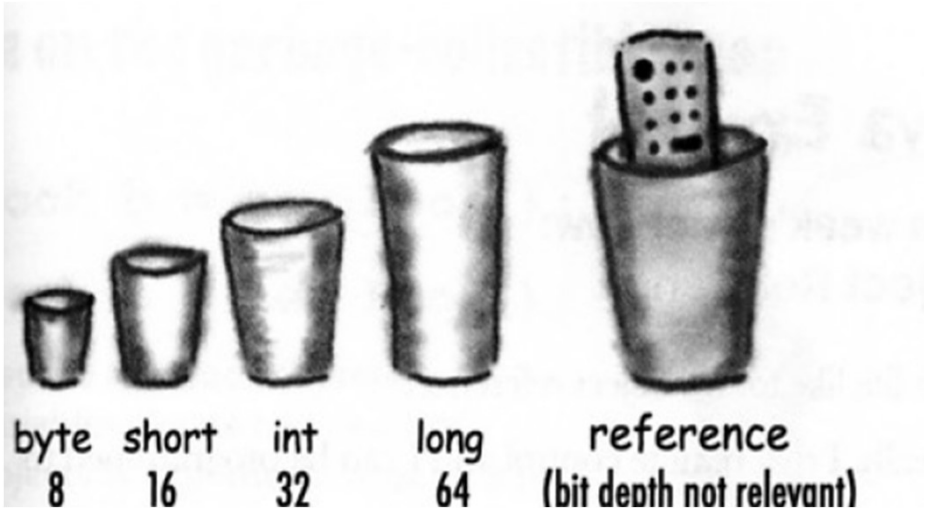
Data types

Note:

- By default, all integers are initialized with 0.
- By default, all floating point numbers are initialized with 0.0
- By default, all characters are initialized with blank space.
- By default, all booleans are initialized with false. This data type represents one bit of information, but its "size" isn't something that's precisely defined.
- By default, all reference types are initialized with null.
- The size of the primitive data types are **fixed** across all operating systems.
- The java supports only signed integers but unsigned characters.

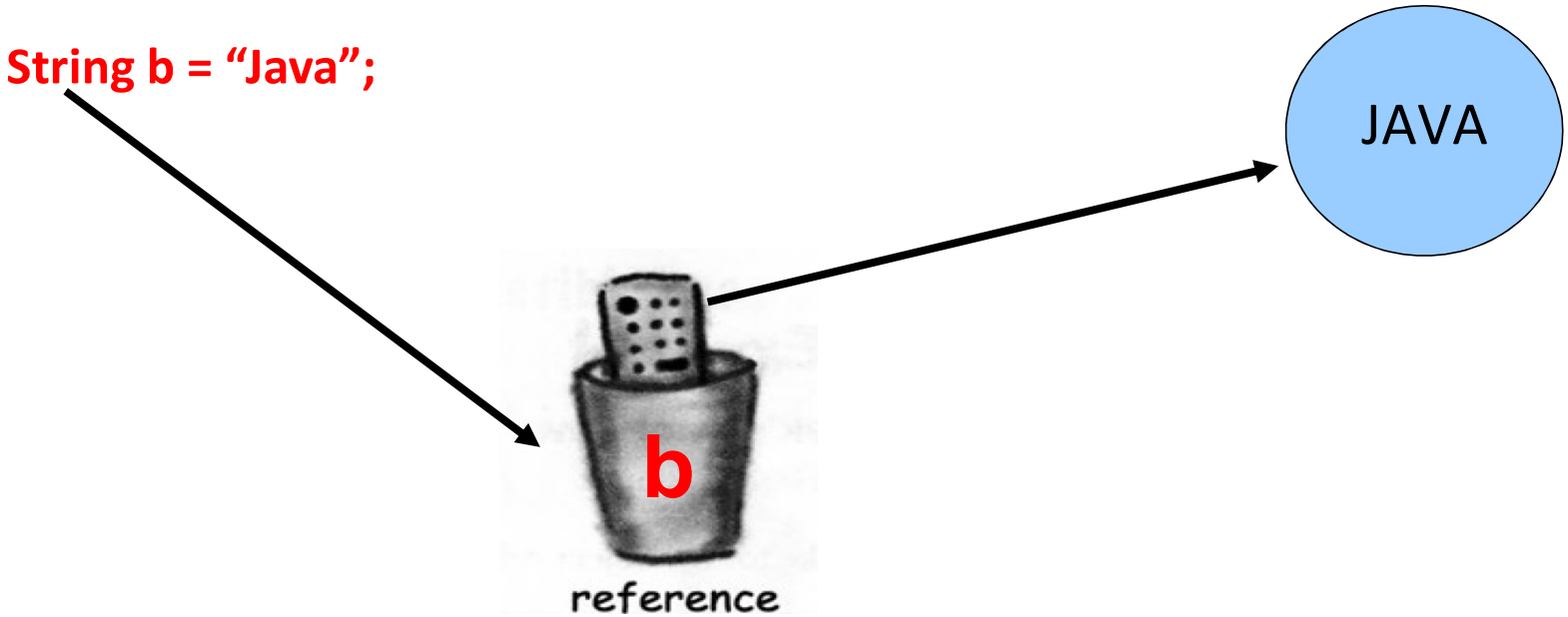
Data types

- **Reference type**
  - A Reference variable holds bits that represents a way to access an object.
  - It doesn't hold the object itself, but it holds something like a pointer, or an address.



Data types

- **Reference type Example:**



Data types

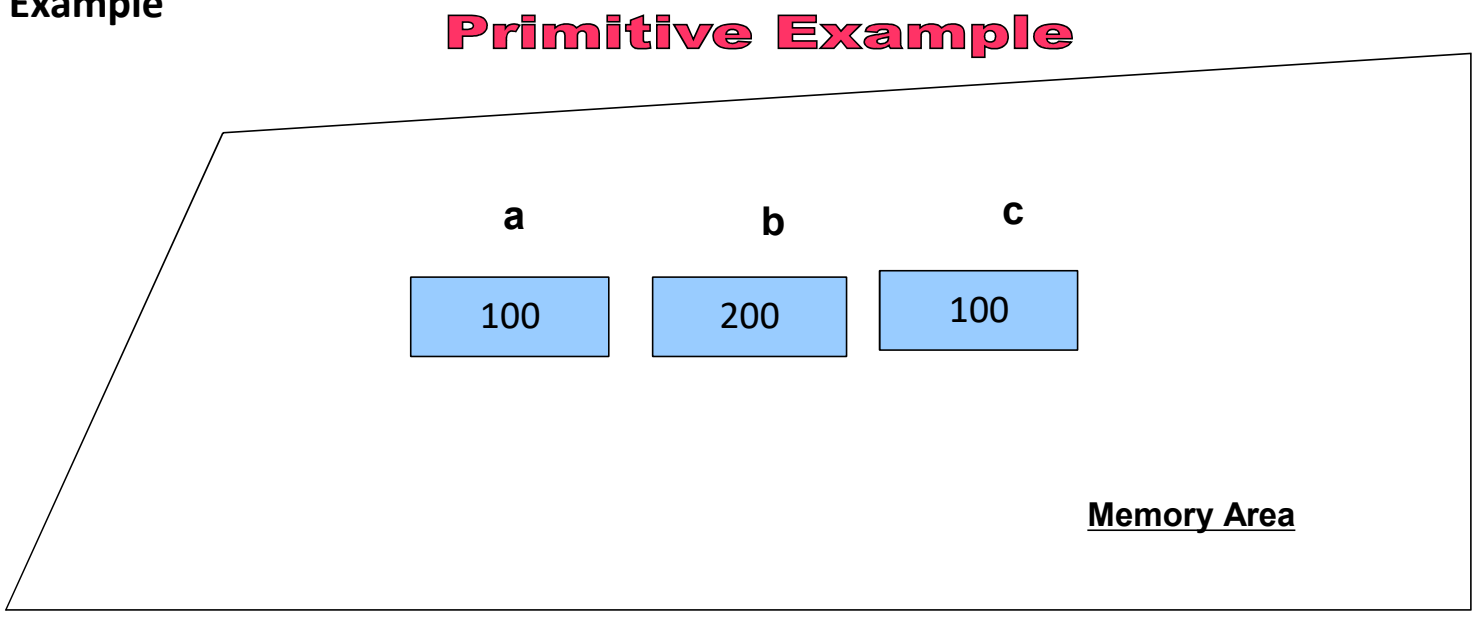
❖ Primitive Vs Reference Type

➢ Primitive Type Example

```
int a = 100;

int b = 200;

int c = a;
```



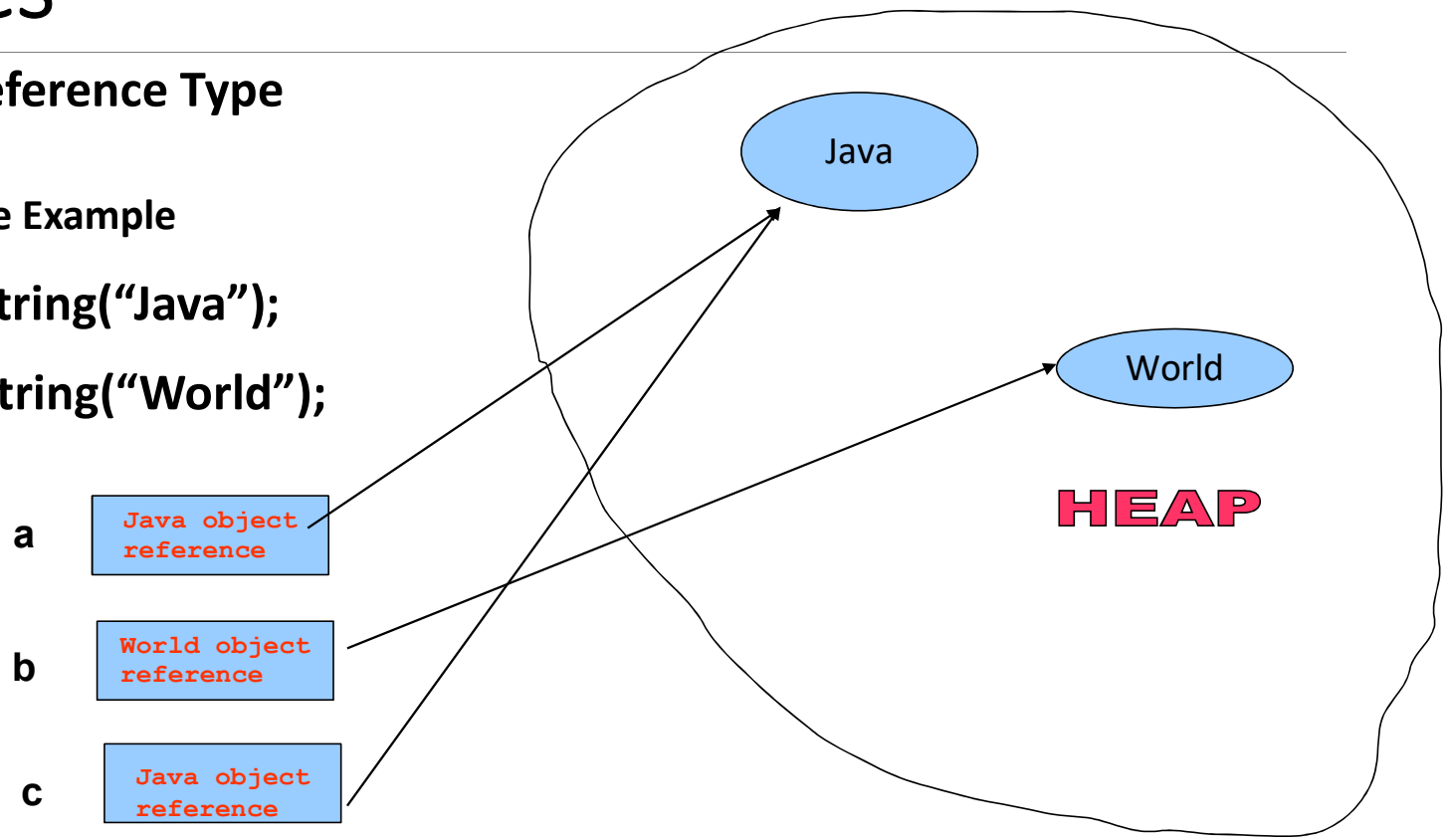
Data types

- **Primitive Vs Reference Type**
  - **Reference Type Example**

String a = new String("Java");

String b = new String("World");

String c = a;



Comments

❖ **Java supports 3 types of comments:**

1) Multline comments (From **C** language)

Syntax:

```
/*
....
....
*/
```

2) Single Line comment (From **C++** language)

Example:

```
//Tomorrow is a holiday
```

3) Java doc comments

This is newly added in JAVA. These comments are used to add description about class as well as methods.

Syntax:

```
/**
*
*/
```

Type casting

- Primitive or Reference type widening does not require explicit cast.
- But, either Primitive or Reference type narrowing requires explicit cast.

Example:

```
float f = 10.10F;
 //int i = f; //Compilation Error saying "Incompatible types"
int i = (int)f; // To avoid compilation error, use type casting
```

Type casting

- Consider the following statements:

```
byte i = 100;
long k = i * 3 + 4;
double d = i * 3.1 + k / 2;
```

Type casting

Conversion Rules

- When performing a binary operation involving two operands of different types, Java automatically converts the operand based on the following rules:
  - If one of the operands is double, the other is converted into double.
  - Otherwise, if one of the operands is float, the other is converted into float.
  - Otherwise, if one of the operands is long, the other is converted into long.
  - Otherwise, both operands are converted into int.

Type casting

- Implicit casting

  double d = 3; (type widening)

- Explicit casting

  int i = (int)3.0; // (type narrowing)

  int i = (int)3.9; // (Fraction part is truncated)

What is wrong? int x = 5 / 2.0;

Reading Input

1. Create a Scanner object

- Scanner input = new Scanner(System.in);

2. Use the methods next(), nextByte(), nextShort(), nextInt(), nextLong(), nextFloat(), nextDouble(), or nextBoolean() to obtain to a string, byte, short, int, long, float, double, or boolean value.

For example,

```
System.out.print("Enter a double value: ");
Scanner input = new Scanner(System.in);
double d = input.nextDouble();
```

Arithmetic Operators

- The arithmetic operators are +, -, *, /, % (Remainder).

Numeric Promotion in Arithmetic Expressions

- Floating point arithmetic is performed if any of the operand type is floating-point type, otherwise, integer arithmetic is performed.

Arithmetic Operators

❖ Numeric Promotion in Arithmetic Expressions

Operand	Operand2	Promoted type
byte	byte	int
short	short	int
char	char	int
int	long	long
int	float	float
long	float	float
double	Any type	double

//Integer Arithmetic

//Floating point Arithmetic

Note : In arithmetic expression, if the operand type is byte, short, or char, then they are automatically promoted to int.

Relational operators

- The relational operators are **<, <=, >, >=**.
- All relational operators are binary operators.
- The evaluation results in a **boolean value**.

Example:

```
System.out.println(1< 2); //true
```

Equality Operators

Equality Operators (==, !=)

- Equality operators are evaluated to **boolean value.**

Example:

- System.out.println(1 == 1); //true
- System.out.println(true == true); //true
- System.out.println((flase == false)); //true
- System.out.println((true == false)); //false
- System.out.println((true != false)); //true
- System.out.println((null == null)); //true

Assignment Operators

There are three types of assignment operators:

- Simple assignment operator

  Exmaple:

  ```
  int a = 10;
  ```

- Chained assignment operator

  Example:

  ```
  int a, b, c;
  a = b = c =10;
  ```

- Compound assignment operator
  - The following are the possible compound assignment operators: +=,-=,*=,/=,%=,&=,|=,^=

  Example:

  ```
  sum += x;          //which is equivalent to sum = sum + x;
  ```

Assignment vs Equality

- **Assignment** sets and/or re-sets the value stored in the storage location denoted by a variable name.
- **Equality** is a relational operator that tests or defines the relationship between two entities.
- In Java, "=" is used for assignment, while "==" is used for comparison.
- int x = 5; // When you use "=", you are assigning a value to a variable.
- if (x == 5) {
  // do something
  } // When you use "==", you are comparing two values to see if they are equal.
- It's important to remember that "==" and "=" are not the same and should not be used interchangeably.

Ternary Operator (?:)

- It is used to decide between two actions.

  syntax:

  <data type> <variable name> = <condition> **?** <expression1> **:** <expression2>;

- If the <condition> is true then <expression1> is evaluated; otherwise, <expression2> is evaluated. The <expression1> and <expression2> must be evaluate to values of compatible types with left hand side declaration.

  Example: Find out max number

```
int a = 10;
int b = 20;
int max = a > b ? a : b;
```

- The ternary operator is equivalent to if-else but additionally returns value.

Boolean logical Operators (&, |, ^, !)

- The boolean logical operators are applicable only for boolean operands, and returning a boolean value.
  - & -> Logical AND
  - | -> Logical OR
  - ^ -> Logical XOR
  - ! -> Logical Complement
- Truth table for boolean logical operators

X	Y	!X	X&Y	X|Y	X^Y
True	True	False	True	True	False
True	False	False	False	True	True
False	True	True	False	True	True
False	False	True	False	False	False

Boolean logical Operators (&, |, ^, !)

- For boolean logical AND, OR, and XOR operators, **both operands are always executed**.

Example:

```
int i = 10;
if( i > 10 & i++ < 20){}        // 'i' is incremented irrespective of first condition.
```

Short-curcuit (or conditional) operators (&&, ||)

- && -> Conditional AND
- || -> Conditional OR

- Truth table for conditional operators:

X	Y	X && Y	X || Y
True	True	True	**True**
True	False	False	**True**
False	True	**False**	True
False	False	**False**	False

Short-curcuit (or conditional) operators (&&, ||)

- **For Conditional AND (&&) operator**, the second operand never evaluated if the first operand is false.
- **For Conditional OR (||) operator**, the second operand is never evaluated if the first operand is true.

Logical Operators (&, |)	Conditional Operators (&&, ||)
The second operand is always evaluated. i.e., both operands are always executed.	The second operand is optionally executed i.e., if the first operand determines the result, then second operand never executed.

Example:

```
int a = 10;
int b = 5;
if((b != 0) && (a % b == 0)){          //Never throws ArithmeticException
System.out.println("No exception");
}
```