

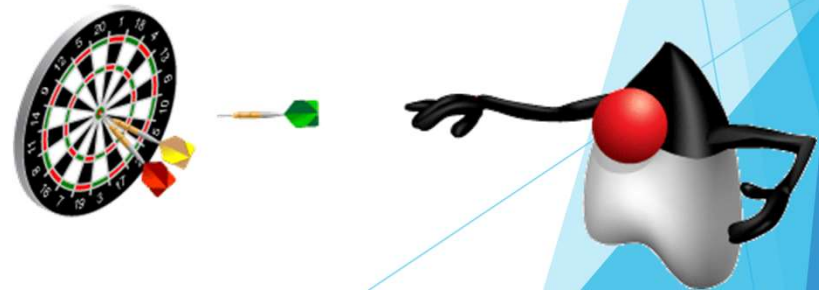
The background features abstract geometric shapes in various shades of blue. On the left, a light blue triangle points downwards. On the right, a complex arrangement of overlapping triangles in light, medium, and dark blue creates a dynamic, layered effect.

# java.lang Package

# Objectives

At the end of this session, you will be able to:

- ▶ Use classes and interfaces in java.lang package.
- ▶ Understand String Buffer Class.
- ▶ Use the Math Class and its methods.
- ▶ Understand Wrapper Classes and use its methods.



# Agenda

- ▶ java.lang package
- ▶ StringBuffer class
- ▶ Math class
- ▶ Wrapper classes

# java.lang

- ▶ The Java package java.lang contains fundamental *classes* and *interfaces* closely tied to the language and runtime system.
- ▶ This package is imported by default in all java programs
- ▶ **Interfaces:**
  - Cloneable
  - Comparable
  - Runnable
- ▶ **Classes:**
  - Object
  - Data Type Wrappers
  - Strings classes
  - System and Runtime
  - Threads
  - Reflection Classes
  - Math
  - Exception, Error, and Throwable
  - Process

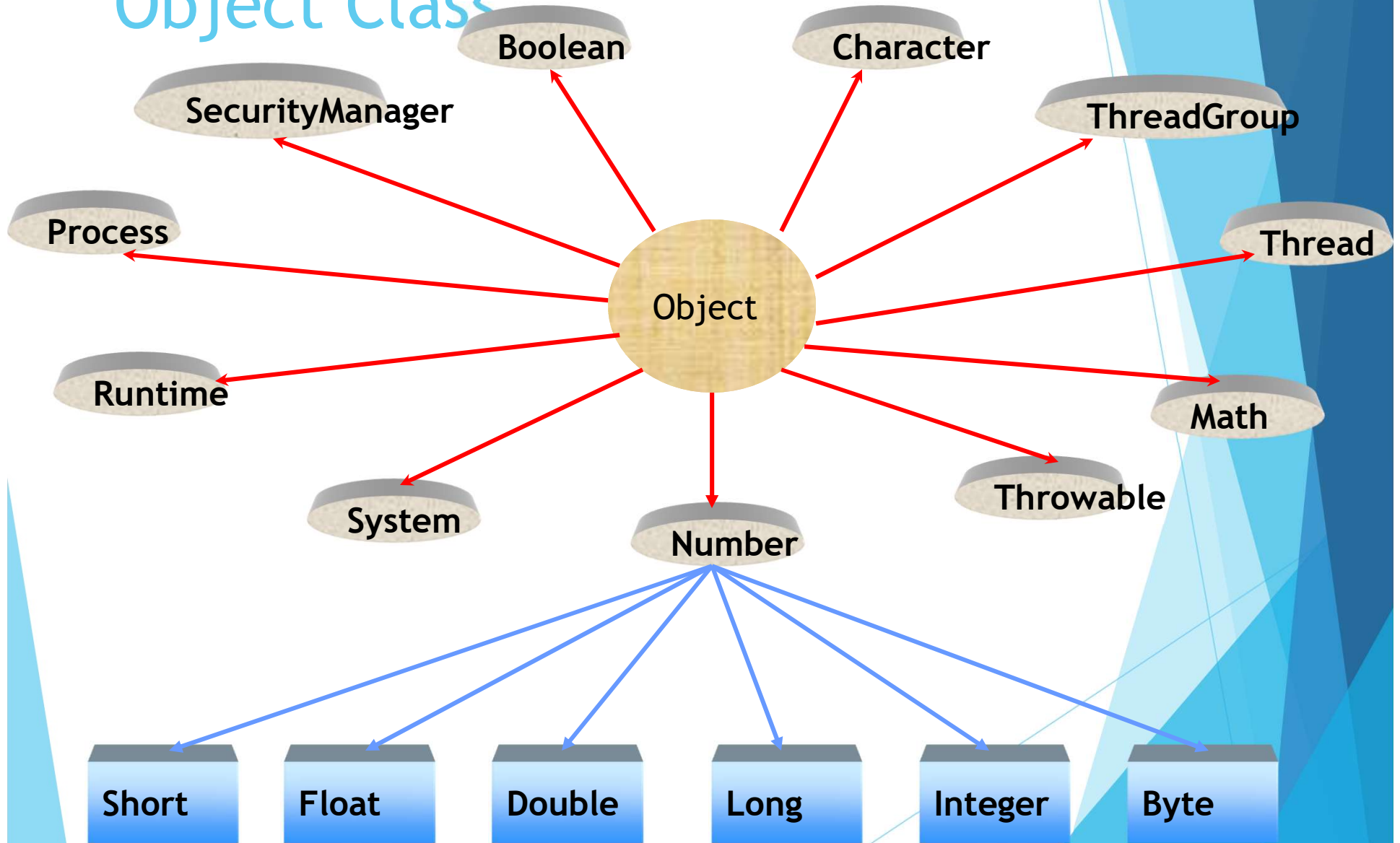
# Object Class

- ▶ Class `java.lang.Object` is the root of the java class hierarchy.
- ▶ Every class has **Object as a superclass**.
- ▶ All objects, including arrays, inherit the methods of this class

## Important methods.

- ▶ **toString()** : returns string representation of the object.
- ▶ **equals()** : compare the object references.
- ▶ **hashCode()** : returns distinct integers for distinct objects
- ▶ **wait(), notify(), notifyAll()**: Thread mechanism

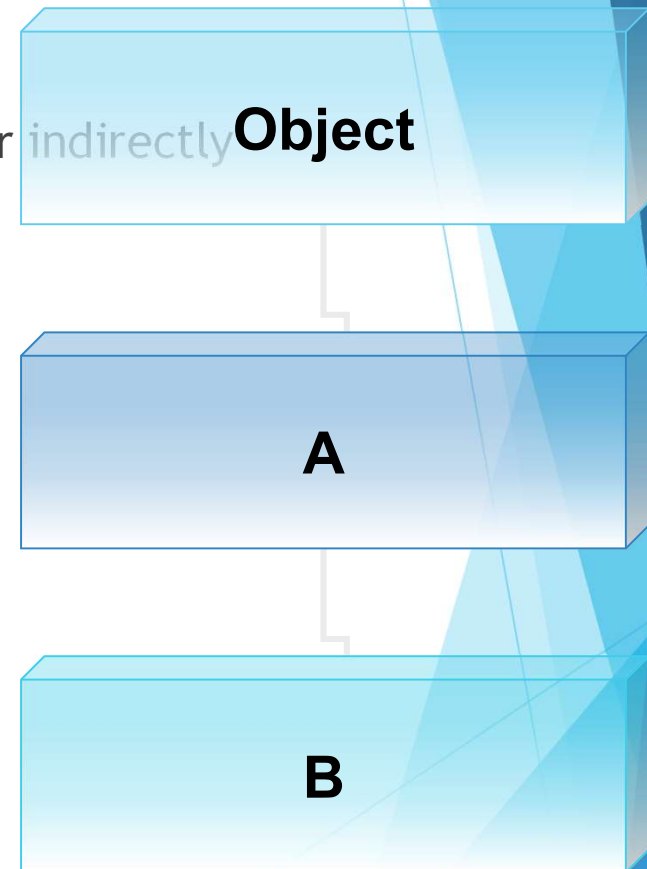
# Object Class



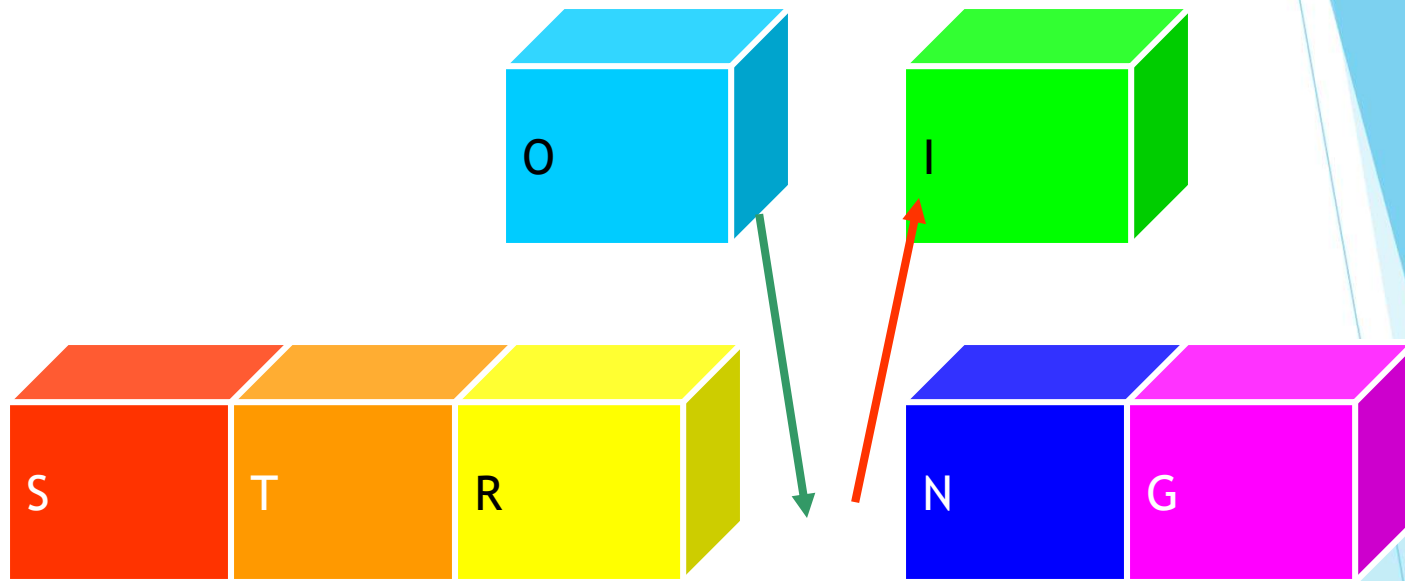
# The Object Class

- ▶ Every class extends Object class directly or indirectly

```
class A
{
    ...
}
public class B extends A
{
    ...
}
```



# Introduction to StringBuffer

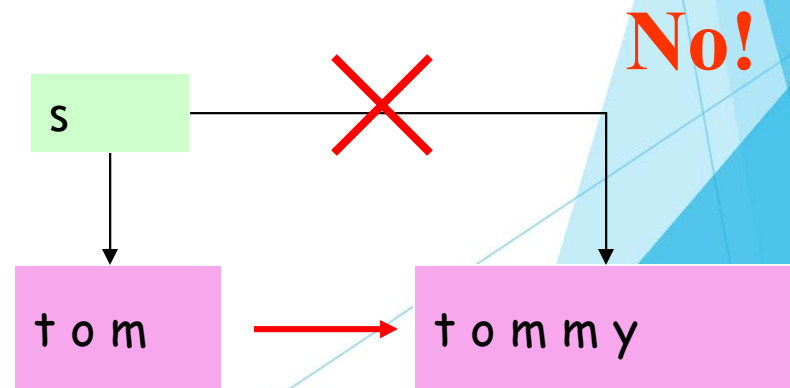




## Limitations of String objects



- We can replace a string or change its value.
- But can we insert or delete characters?

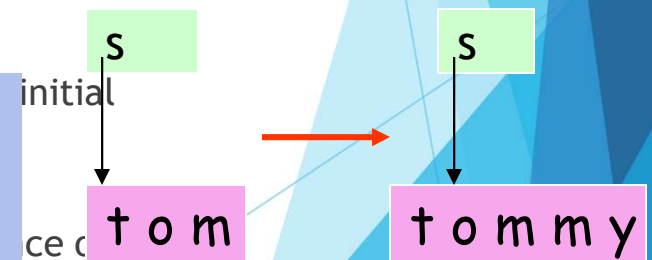


# String Buffer

- ▶ A **string buffer** is like a string except that it is *mutable*, i.e. we can add to the string.
- ▶ It represents **growable** and **writable** character sequences
- ▶ There are three **StringBuffer constructors** available:
  - **StringBuffer()**: no characters and initial capacity of 16 characters.

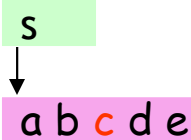

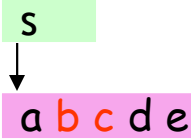

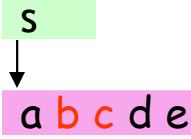

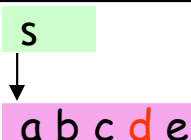



```
StringBuffer s= new  
StringBuffer("tom");  
s.append("my");
```

characters as the string argument.

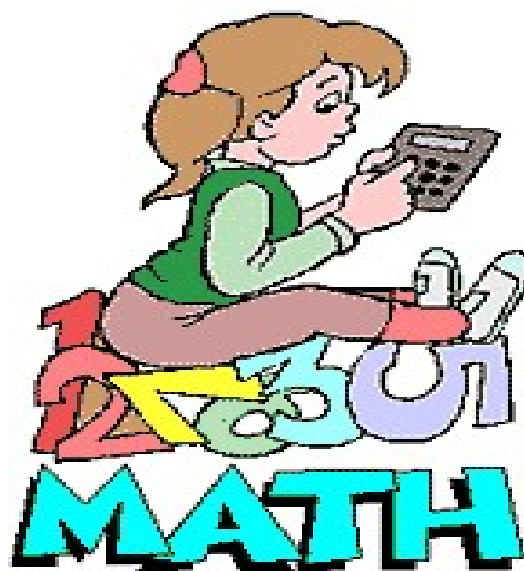


## String Buffer Methods

*StringBuffer* s= new  
*StringBuffer*("abcde");

❶	s.append(2,"xyz");		
❷	s.delete(1,3);		
❸	s.replace(1,3,"xyz");		
❹	s.setCharAt(3,'x');		
❺	s.reverse();		

## Math Class



## Math Class

- ▶ The **Math class** is **final** so you **cannot inherit** this class.
- ▶ All the **methods** defined in the Math class are **static**. Such classes are commonly referred to as utility classes
- ▶ Also, the Math class has a private constructor, so you cannot instantiate it.

### The Math class has the following methods:

- ▶ **ceil()**: returns the smallest double value that is not less than the argument and is equal to a mathematical integer.

- **Math.ceil(5.4)** // gives 6
- **Math.ceil(-6.3)** // gives -6

## Math Class

- ▶ **floor()**: returns the largest double value that is not greater than the argument and is equal to a mathematical integer.

- `Math.floor(5.4) // gives 5`
- `Math.floor(-6.3) // gives -7`

- ▶ **max()**: takes two numeric arguments and returns the greater of the two. It is overloaded for int, long, float, or double arguments.

- `x = Math.max(10,-10); // returns 10`

- ▶ **min()**: takes two numeric arguments and returns the smaller of the two. It is overloaded for int, long, float, or double arguments.

- `x = Math.min(10,-10); // returns -10`

## Math Class

- ▶ **random()**: returns a double value with a positive sign, greater than or equal to 0.0 and less than 1.0.

- ▶ **abs()**: returns the absolute value of the argument.

- `Math.abs(-33)`    `// returns 33`

- ▶ **round()** : returns the integer closest to the argument .If the number after the decimal point is less than 0.5, `Math.round()` returns the same result as `Math.floor()`. In all the other cases, `Math.round()` works the same as `Math.ceil()`.

- `Math.round(-1.5)`    `// result is -1`

- `Math.round(1.8)` `// result is 2`

# Math Class

## ▶ Trigonometric functions:

- `sin()`, `cos()`, and `tan()` methods return the sine, cosine, and tangent of an angle, respectively.
- The argument is the angle in radians.
- Degrees can be converted to radians by using `Math.toRadians()`.
- ```
x = Math.sin(Math.toRadians(90.0)); // returns 1.0
```
- 

## ▶ `sqrt()` : returns the square root of an argument of type double. :

- ```
x = Math.sqrt(25.0); // returns 5.0
```

**Note:** If you pass a negative number to the `sqrt()` function, it returns NaN ("Not a Number").



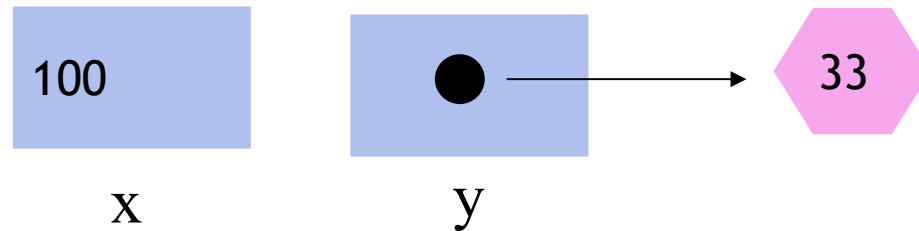
## Wrapper Class

- ▶ Some times we need to treat primitives like int, char, float values as Objects
- ▶ Java provides Wrapper Classes for each primitive type which wraps the primitive value as an Object

Primitive Type	Wrapper Class
byte	Byte
short	Short
char	Character
int	Integer
long	Long
float	Float
double	Double
boolean	Boolean



## Wrapper Class



- ▶ Difference between a primitive data type and an object of a wrapper class:

```
int x = 100;
```

```
Integer y = new Integer(33); // Before  
Java 5.0
```

- `x` is a variable that holds a value
- `y` is an object variable that holds a reference to an object.

## Wrapper Classes

- ▶ All of the primitive wrapper classes in Java are **immutable** i.e. once an object is created, the wrapped primitive **value cannot be changed**
- ▶ The Wrapper classes do not contain a no-argument constructor
- ▶ All the wrapper classes except Character have two constructors

1) takes the primitive value.

```
Integer i1 = new Integer(50);
```

2) takes the String representation of the value.

```
Integer i2 = new Integer("50");
```

- ▶ The Character class constructor takes a char type element as an argument.

```
Character c = new Character('A');
```

## AutoBoxing and UnBoxing

- ▶ Boxing is the process of converting primitive values into objects of corresponding wrapper class

```
Integer y = 100; // Java 5 supports AutoBoxing  
Integer y = new Integer(100); //Prior to Java 5
```

- ▶ UnBoxing is the process of converting objects of wrapper type to corresponding primitive values

```
int x = y ; // Java 5 supports AutoUnBoxing  
int x = y.valueOf(); //Prior to Java 5
```

## Significance of Wrapper Classes

- Wrapper classes have a lot of useful methods

Examples:

```
Character.toLowerCase(ch)
```

```
Character.isLetter(ch)
```

- A common translation is converting a string to a numeric type such as an int

Examples:

```
String s = "1000";
```

```
int i = Integer.parseInt(s);
```

## Summary

In this session, we have covered

- ▶ `java.lang` package
- ▶ `StringBuffer` class
- ▶ `Math` class
- ▶ Wrapper classes

**Thank you**

