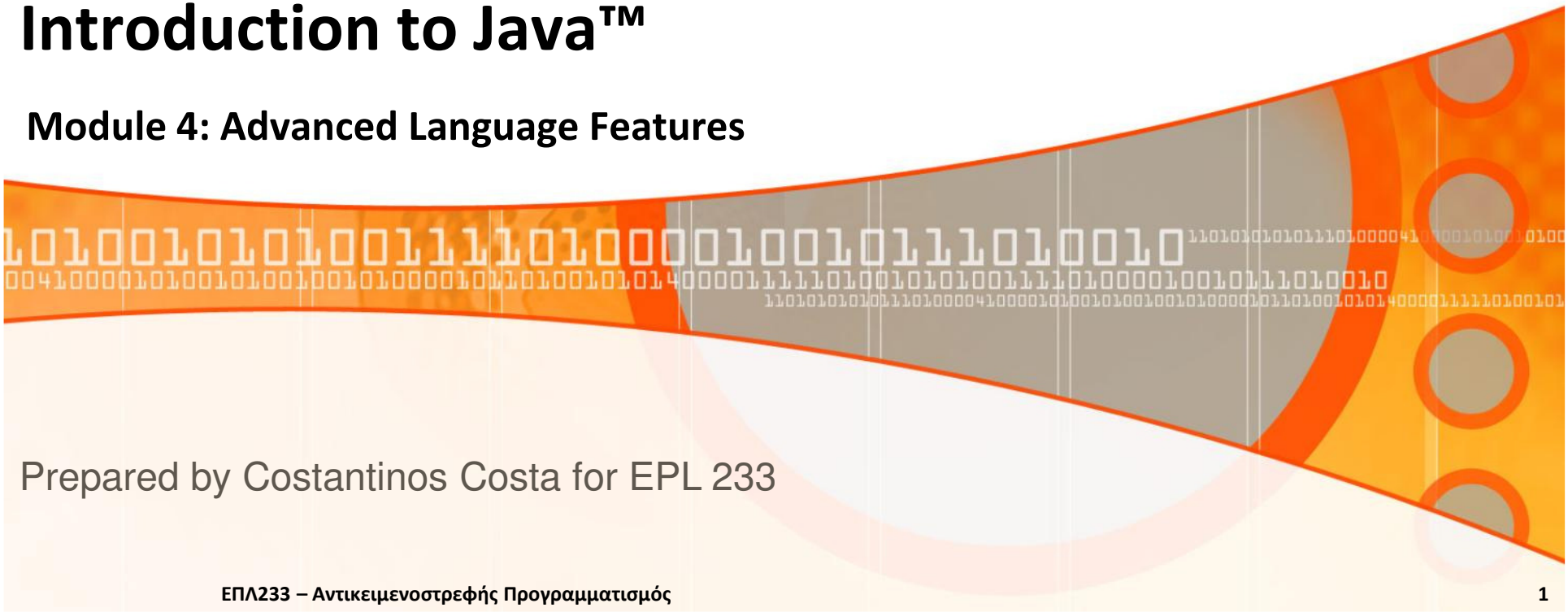
Introduction to Java™

Module 4: Advanced Language Features



Prepared by Costantinos Costa for EPL 233

Constructors

- **Constructor is the method called when an object is initialized with the new keyword**
- **Constructors can also be overridden (they usually are)**
- **Constructors can invoke constructors from the superclass**

Example:

```
public class redCircle extends Circle {……
        //constructor
        public redCircle(int x, int y,int radius) {
                super(x,y,radius);
                color=Color.red;
        }
}
```

Static variables & methods

- Static methods
  - No need to create an instance of the class containing the method to use it

    Example:

    double x = Console.readDouble();

    double x = Math.pow(3, 0.1);

    This works because the method readDouble of the Console class and the method pow of the Math class are declared as static

    I.e. public static double readDouble() {……}

Static variables & methods

- Static variables
  - No need to create an instance of the class to access them

Example:

```
public class defaults {
   public static String hostname="java.sun.com"
……}

public class anotherClass {
  ……
  System.out.println("The hostname is " +
defaults.hostname);
  ……}
```

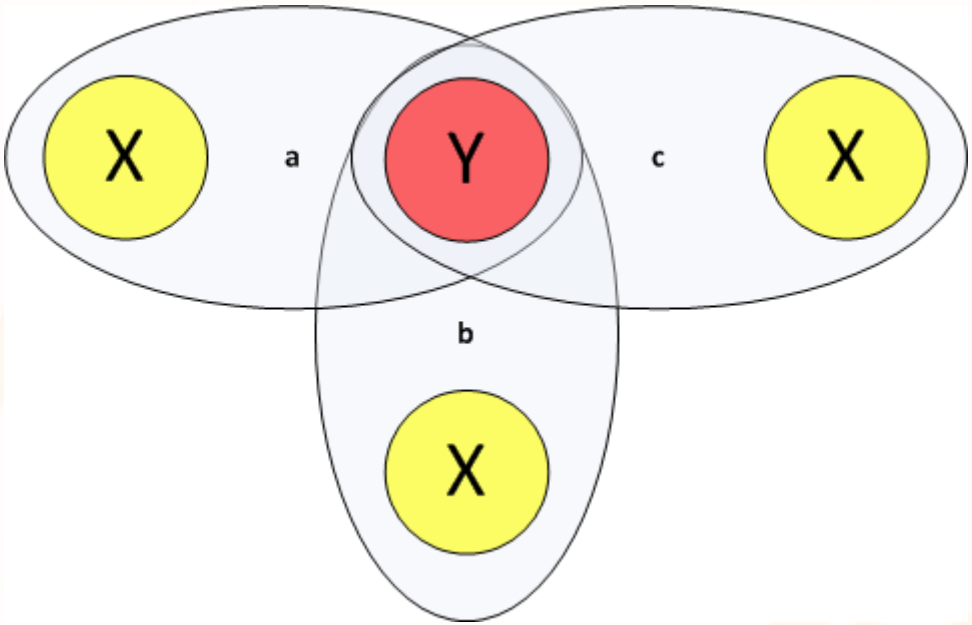
Static variables & methods

- We need to create a new object to access data and methods of the specific class.
- What if we need to access those data and methods without create an object ??
- Use **static** ☺
- Static variables and methods are called **class-data**, **class-methods**.

Static variables & methods

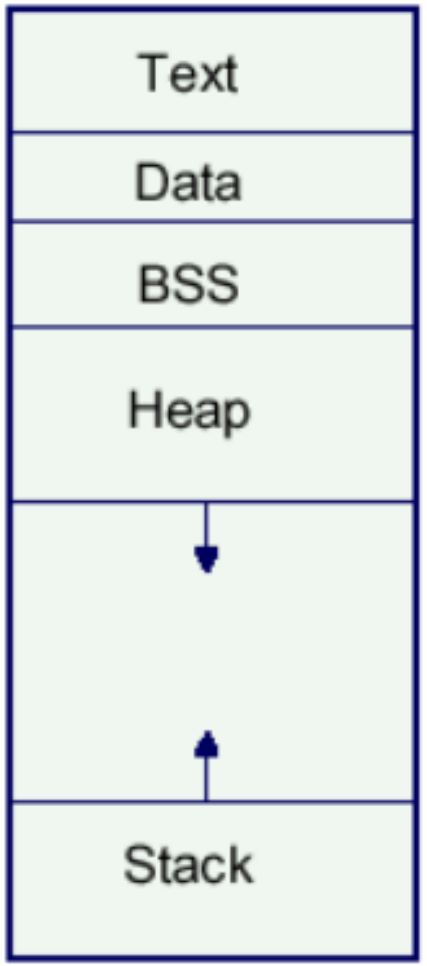
```
public class WithStaticData {
    static int y;
    int x;

    public static void main(String[] args) {
        WithStaticData a = new WithStaticData();
        WithStaticData b = new WithStaticData();
        WithStaticData c = new WithStaticData();
    }
}
```



Memory

- **Text:** **binary code** of the process's program
- **Data**: **constants** of the program
- **BSS** (Block Started by Symbol):**Global** and **static** variables
- **Stack**: **stack**, local variables
- **Heap**: **dynamic memory**



Final classes, variables and methods

- A final class cannot become a parent class
  Example: final class Card{ …… }
- Specific methods in a class can be declared as final. A final method cannot be overridden
  Example: public final void doSomethind()
  {……}
- A final variable cannot change value
  Example: public final int x=15;
- Using final improves performance

Wrapper classes

- An instance of the Double class wraps the double type, Integer the int type and so on…

Example:

Suppose we need a vector of **Double**. Simply adding numbers to the vector won't work:

- Vector v = new Vector();
- v.addElement(3.14); //ERROR

The floating-point number 3.14 is not an object. Here we can use the **Double** wrapper class to create a Double object and add it to the vector:

v.addElement(new Double(3.14));

Working with Strings

Java.lang.String

- Create:
  String x = new String(“a string”);
- Concat:
  x=x+”, another string”; (x=“a string, another string”)
- Length:
  int stringLength = x.length();
- Comparing
  if (x.equals(“a string”))
  if (x.compareTo(“a string”))

Working with Strings

- **Other useful methods:**
  - **indexOf(String)**
    - Returns the index within this string of the first occurrence of the specified character.
  - **replace(char, char)**
    - Returns a new string resulting from replacing all occurrences of oldChar in this string with newChar
  - **startsWith(String)**
    - Tests if this string starts with the specified prefix.
  - **trim()**
    - Removes white space from both ends of this string.
- **Check the API documentation for the complete list of functions**

Working with Strings

- Java provides three classes for working with Strings
  - *String*, *StringBuffer* and *StringBuilder*
- Class String is used for strings that remain constant (their value doesn't change)
- Class StringBuffer and *StringBuilder* is used for strings that may change
  - E.g. Reading the contents of a text file
  - Check out the Java API for StringBuffer and *StringBuilder*
- Using String is more efficient when our strings remain unchanged as they are constants that can be jointly used by other code in our program
- StringBuffer and *StringBuilder* is more efficient when we have changing strings as we create only one object
- StringBuffer is synchronized, *StringBuilder* is not.

Concatenating: + Operator vs StringBuffer vs StringBuilder

```
public class StringConcatanations {
  public static int MAX_ITER = 100;
  public static void main(String[] args) {
  for (int i = 1; i <=5; i++) {
    MAX_ITER=(int)Math.pow(10.0,i);
    concatenate();
    concatenateWithStringBuffer();
    concatenateWithStringBuilder();
          }
  }
  public static void concatenate() {
    System.out.println("Concatanating using the + operator");
    String s1 = "";
    long s1Time = getNanoTime();
    for(int i=0;i<MAX_ITER;i++) {s1 = s1 + "abc"; }
    long e1Time = getNanoTime();
    System.out.println("Time: " + (e1Time - s1Time));
  }
  public static void concatenateWithStringBuffer() {
    System.out.println("Concatanating using StringBuffer");
    StringBuffer sb = new StringBuffer();
    long s2Time = getNanoTime();
    for(int i=0;i<MAX_ITER;i++) {sb.append("abc"); }
    long e2Time = getNanoTime();
    System.out.println("Time: " + (e2Time - s2Time));
  }
```

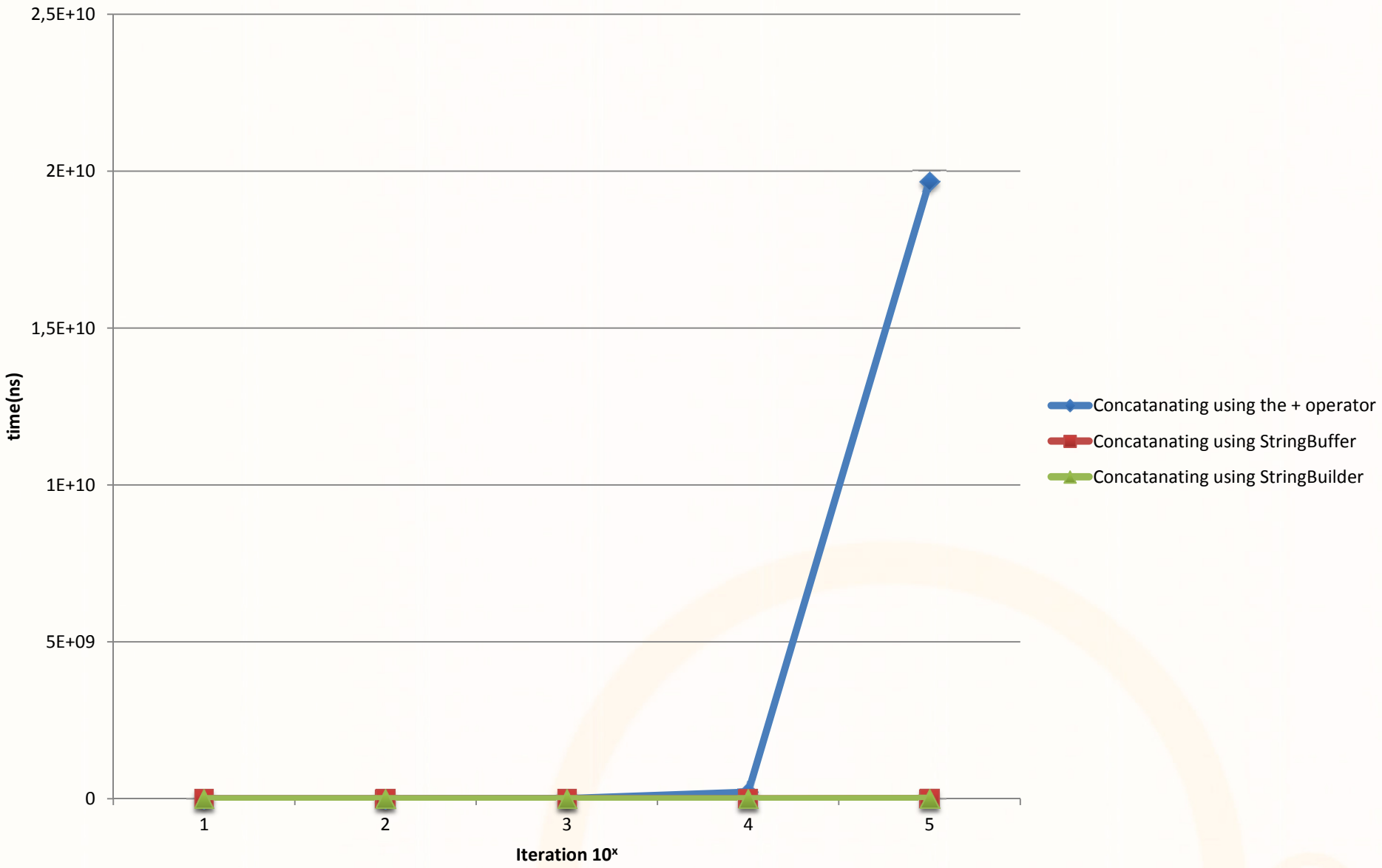
Concatenating: + Operator vs StringBuffer vs StringBuilder

```
  public static void concatenateWithStringBuilder() {
    System.out.println("Concatanating using StringBuilder");
    StringBuilder sBuilder = new StringBuilder();
    long s3Time = getNanoTime();
    for(int i=0;i<MAX_ITER;i++) {
      sBuilder.append("abc");
    }
    long e3Time = getNanoTime();
    System.out.println("Time: " + (e3Time - s3Time));
  }

  public static long getNanoTime() {
      return System.nanoTime();
  }

}
```

Concatenating: + Operator vs StringBuffer vs StringBuilder



Working with Strings

```
class SameString {
    public static void main(String[] args) {
        String s1 = “dog”;
        String s2 = “It’s a dog’s life”;
        String s3 = “dog”;
        if (s1 == s2) System.out.println(“s1 == s2”); // FALSE
        if (s1 == s3) System.out.println(“s1 == s3”); // TRUE
        if (s1 == “dog”)  System.out.println(“s1 == \“ dog \””);
    //TRUE
        String doggy = new String(s1);
        if (s1 == doggy) System.out.println(“s1 == doggy”);
    //FALSE
    }
}
```

Example-Exercise

- Checking whether a string is a palindrome.
  - *Palindrome* a string that reads the same forward and backward.
  - E.g
    - Level, civic ,“νίψον ανομήματα μη μόναν οψιν”

Example-Exercise

- Checking whether a string is a palindrome. Without using a loop !!!
  - *Palindrome* a string that reads the same forward and backward.
  - E.g
    - Level, civic ,“νίψον ανομήματα μη μόναν οψιν”