Java SE 5: Newly Introduced Language Features

Sang Shin
Michèle Garoche
www.javapassion.com
"Learn with Passion!"

Java SE 5 Design Themes

- Focus on quality, stability, compatibility
  - Most Java software now runs over Java SE 5
- Support a wide range of application styles
  - "From desktop to data center"
- Big emphasis on scalability
  - Exploit big heaps, big I/O, big everything
- Continue to deliver great new features
  - Maintaining portability and compatibility
- Ease of development
  - Faster, cheaper, more reliable

Language Changes

Java Language Changes

- JDK 1.0
  - Initial language, very popular
- JDK1.1
  - Inner classes, new event model
- JDK 1.2, 1.3
  - No changes at language level
- JDK 1.4
  - Assertions (minor change)
- JDK 5.0 (Java SE 5)
  - Biggest changes to language since release 1.0

Major New Language Features

- Generics (will be covered in another presentation)
- Annotation (will be covered in another presentation)
- Autoboxing/Unboxing
- Enhanced for loop (“foreach”)
- Type-safe enumerations
- Varargs
- Static import

Autoboxing & Unboxing

Autoboxing/Unboxing of Primitive Types

- Problem: (pre-Java SE 5)
  - Conversion between primitive types and wrapper types need to be done by programmer
  - Example: you need manually convert a primitive type to a wrapper type before adding it to a collection

```
int i = 22;
List l = new LinkedList();

// Convert int to Integer since an item in a Collection
// cannot be a primitive type
l.add(new Integer(i));
```

Autoboxing/Unboxing of Primitive Types

- Solution: Let the compiler do it (in Java SE 5)

```
Byte byteObj = 22;      // Autoboxing conversion
int i = byteObj         // Unboxing conversion

ArrayList<Integer> al = new ArrayList<Integer>();
al.add(22);  //  Autoboxing conversion
```

Enhanced “for” Loop

Enhanced for Loop (foreach)

- Problem: (pre-Java SE 5)
  - Iterating over collections is tricky
  - Iterator is error prone
    (Can occur three times in a for loop)
- Solution: Let the compiler do it (Java SE 5)
  - New for loop syntax
    ```
    for (Type variable: collection)
    ```
  - Works for both Collections and arrays

Enhanced for Loop Example

- Old code (pre-Java SE 5)

```
void cancelAll(Collection c) {
  for (Iterator i = c.iterator(); i.hasNext(); ){
    TimerTask task = (TimerTask)i.next();
    task.cancel();
  }
}
```

- New Code (Java SE 5)

```
void cancelAll(Collection<TimerTask> c) {
  for (TimerTask task : c)
    task.cancel();
}
```

Type-safe Enumerations

Type-safe Enumerations

- Problem: If you wanted to define an enumeration you either: (pre-Java SE 5)
  - Defined a bunch of integer constants
  - Followed one of the various “type-safe enum patterns”
- Issues of using Integer constants (pre-Java SE 5)
  - public static final int SEASON_WINTER = 0;
  - Public static final int SEASON_SUMMER = 1;
  - Not type safe (any integer will pass)
  - No namespace (SEASON_*)
  - Brittleness (how do I add a value in-between?)
  - Printed values uninformative (prints just int values)

Type-safe Enumerations in Java SE 5

- Solution: New type of class declaration
  - enum type has public, self-typed members for each enum constant
  - New keyword, enum

```
// Definition of enum class
public enum FootballScore {
        TOUCHDOWN( 6 ), FIELDGOAL( 3 ), TWOPOINTS( 2 ),
                SAFETY( 2 ), EXTRAPOINT( 1 );
        //...
}

// Usage of enum class
FootballScore[] yourScores = {
        FootballScore.FIELDGOAL,
        FootballScore.TOUCHDOWN,
        FootballScore.FIELDGOAL
};
```

Varargs

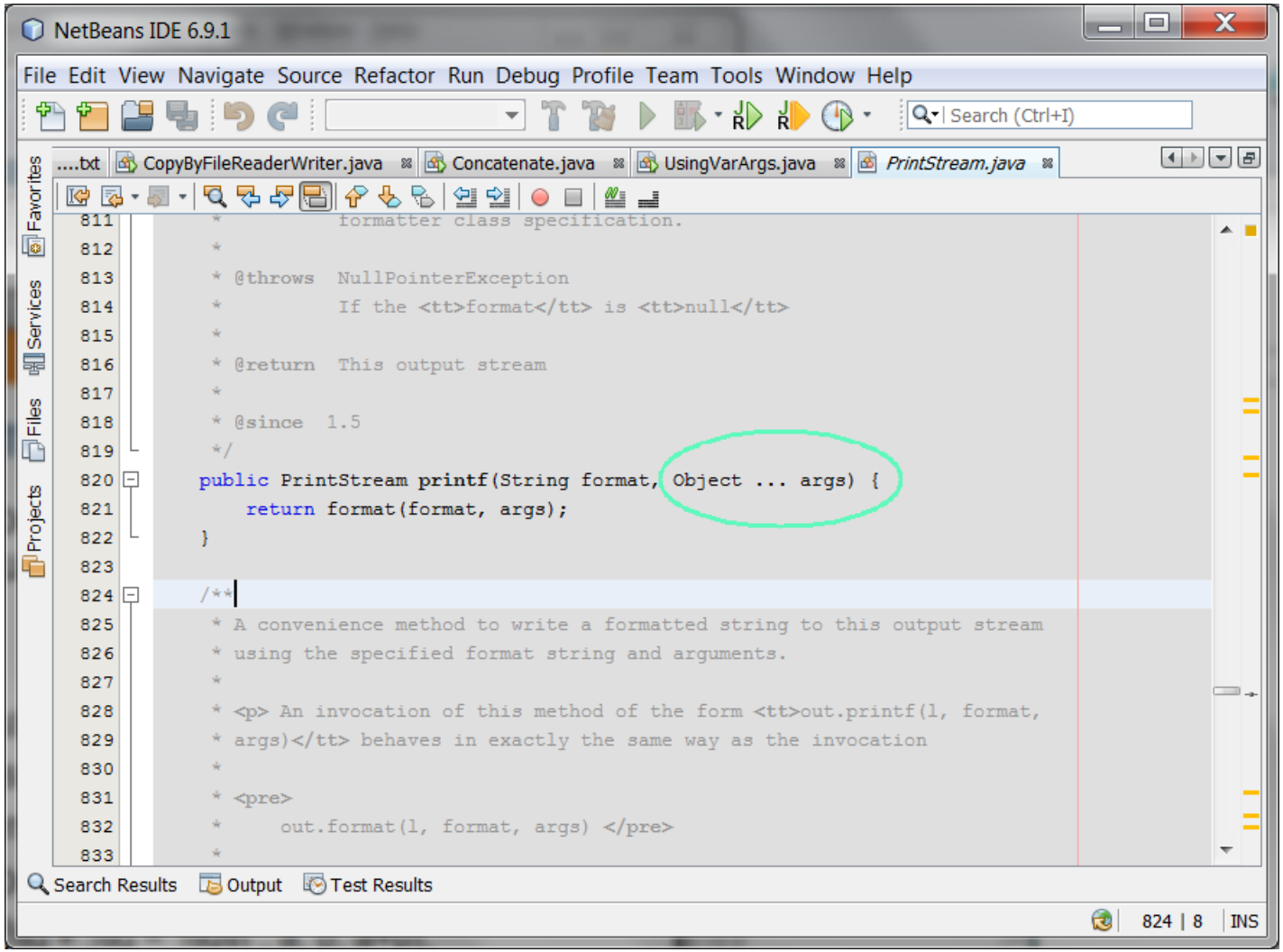
Varargs

- Problem: (in pre-Java SE 5)
  - To have a method that takes a variable number of parameters
  - Can be done with an array, but caller has to create it first
  - Look at *java.text.MessageFormat*
- Solution: Let the compiler do it for you (Java SE 5)
  - public static String format (String fmt, *Object...* args);
  - Java now supports printf(...)

Varargs examples

- APIs have been modified so that methods accept variable-length argument lists where appropriate
  - Class.getMethod
  - Method.invoke
  - Constructor.newInstance
  - Proxy.getProxyClass
  - MessageFormat.format
- New APIs do this too
  - System.out.printf("%d + %d = %d\n", a, b, a+b);

printf(String format, Object … args)



Static Imports

Static Imports

- Problem: (pre-Java SE 5)
  - You have to fully qualify every static referenced from external classes
- Solution: New import syntax (Java SE 5)
  - `import static TypeName.Identifier;`
  - `import static Typename.*;`
  - Also works for static methods and enums

    e.g `Math.sin(x)` becomes `sin(x)`

Formatted I/O

printf

- printf is popular with C/C++ developers
  - Powerful, easy to use
- Finally adding printf to Java SE 5 (using varargs)

```
out.printf("%-12s is %2d long", name, l);
out.printf("value = %2.2F", value);
```

Scanner Class

- A simple text scanner which can parse primitive types and strings using regular expressions.
- A Scanner breaks its input into tokens using a delimiter pattern, which by default matches whitespace. The resulting tokens may then be converted into values of different types using the various next methods.
- Example

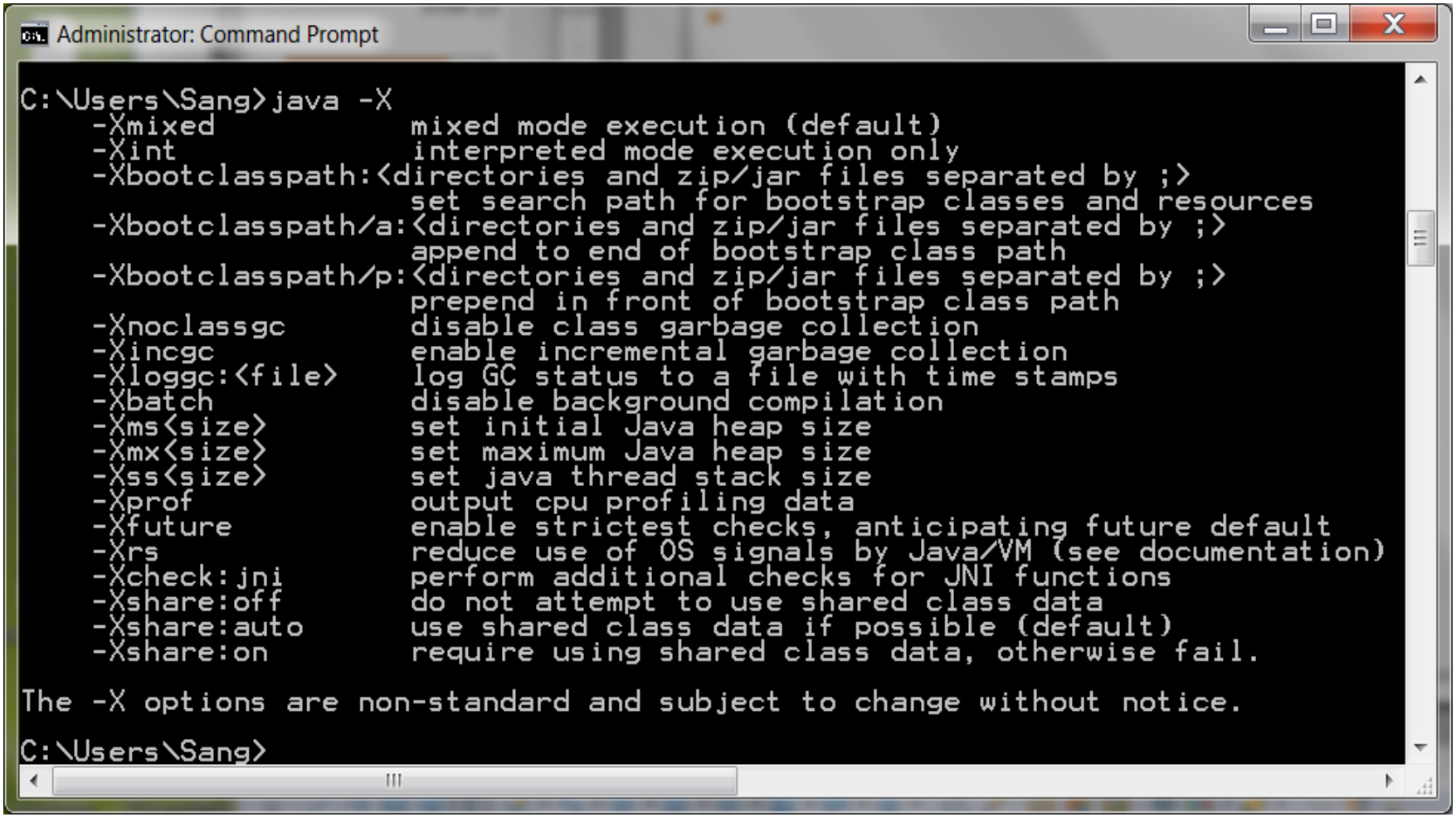
```
Scanner sc = new Scanner(System.in);
int i = sc.nextInt();
```

Virtual Machine

Class Data Sharing

- Improved startup time
  - Especially for small applications
  - up to 30% faster
- Reduced memory footprint
- During JRE installation, a set of classes are saved into a file, called a "shared archive"
- During subsequent JVM invocations, the shared archive is memory-mapped in
- -Xshare:on, -Xshare:off, -Xshare:auto, -Xshare:dump

Class Data Sharing



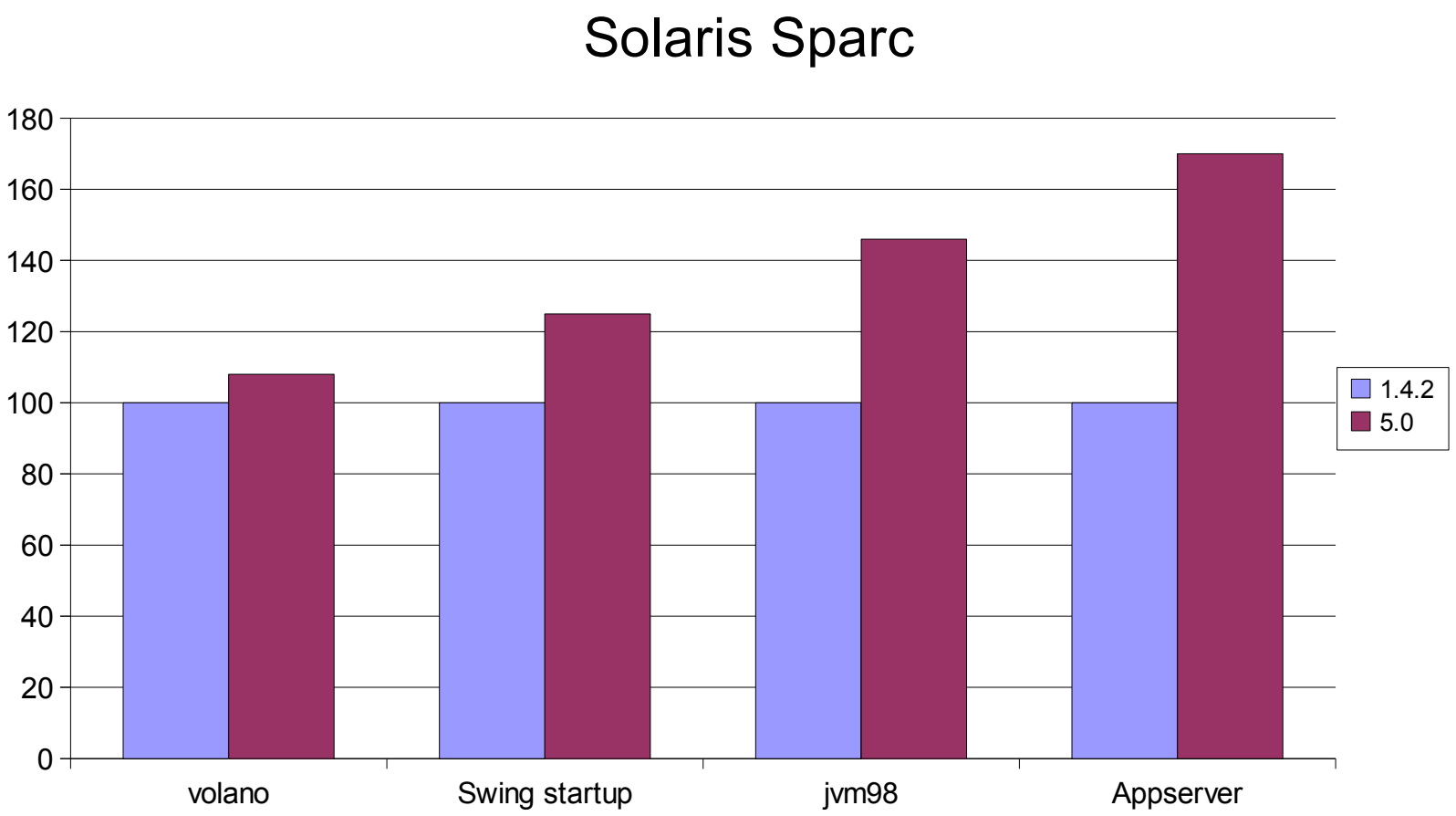
Server Class Machine

- Auto-detected
  - Application will use Java HotSpot Server VM
  - Server VM starts slower but runs faster than Client VM
- 2 CPU, 2GB memory (except Windows)
  - Uses server compiler
  - Uses parallel garbage collector
  - Initial heap size is 1/64 of physical memory up to 1GB
  - Max heap size is 1/4 of physical memory up to 1GB

JVM Self Tuning (Ergonomics)

- Maximum pause time goal
  - -XX:MaxGCPauseMillis=<nnn>
  - This is a hint, not a guarantee
  - GC will adjust parameters to try and meet goal
  - Can adversely effect application throughput
- Throughput goal
  - -XX:GCTimeRatio=<nnn>
  - GC Time : Application time ratio = 1 / (1 + nnn)
  - e.g. -XX:GCTimeRatio=19 (5% of time in GC)

Performance Improvement



Thank you!

Check JavaPassion.com Codecamps!
http://www.javapassion.com/codecamps
"Learn with Passion!"