CMR
INSTITUTE OF
TECHNOLOGY

USN | | | | | | | | | | |

Internal Assessment Test 3– Apr. 2025

Sub:	Advanced Java& J2EE							Sub Code:	22MCA341
Date:	08/4/2025	Duration:	90 min's	Max Marks:	50	Sem:	III	Branch:	MCA

Note : Answer FIVE FULL Questions, choosing ONE full question from each Module

		MARKS	OBE	
			CO	RBT
	PART I			
1	Explain the lifecycle of a servlet in detail.	10	CO1	L2
	OR			
2.	Write a JAVA Servlet Program to implement a dynamic HTML using Servlet (user name and Password should be accepted using HTML and displayed using a Servlet)	10	CO2	L4
	PART II			
3	Describe the classes and interfaces of javax.servlet package.	10	CO1	L2
	OR			
4.	Write a Servlet program to read data from a HTML form (gender data from radiobuttons and colours data from check boxes) and display	10	CO2	L4
	PART III			
5	Explain how do you handle Http Requests and response with an example program?	10	CO1	L2
	OR			
6	Write a JAVA Servlet Program using cookies to remember user preferences.	10	CO2	L4
	PART IV			
7	What is JSP? What are different types of tags in JSP demonstrate with an example.	10	CO1	L2
	OR			
8.	Write a java servlet problem for auto page refresh.	10	CO2	L4
	PART V			
9.	List and explain any 5 collection algorithms. Demonstrate various algorithms with an example program.	10	CO1	L2
	OR			
10.	Explain the need of generic collections	10	CO1	L2

1. Explain the lifecycle of a servlet in detail.

Init() :

- The init method is designed to be called only once.
- It is called when the servlet is first created, and not called again for each user request. So, it is used for one-time initializations, just as with the init method of applets.
- The servlet is normally created when a user first invokes a URL corresponding to the servlet, but you can also specify that the servlet be loaded when the server is first started.
- The init() method simply creates or loads some data that will be used throughout the life of the servlet.

The init method definition looks like this:

```
public void init() throws ServletException
{
// Initialization code...
}
```

Service() Method:

Theservice() method is the main method to perform the actual task.

•Theservlet container (i.e. web server) calls the service() method to handle requests coming from the client(browsers) and to write the formatted response back to the client.

• Each time the server receives a request for a servlet, the server spawns a new thread and calls service.

The service() method checks the HTTP request type (GET, POST, PUT, DELETE, etc.) and calls doGet, doPost, doPut, doDelete, etc. methods as appropriate.

Syntax:

```
public void service(ServletRequest request, ServletResponse response) throws ServletException, IOException
{
}
```

The doGet() Method

- A GET request results from a normal request for a URL or from an HTML form that has no
  METHOD specified and it should be handled by doGet() method.
  Syntax:
  public void doGet(HttpServletRequest request, HttpServletResponse response)

```
throws ServletException, IOException
{
        // Servlet code
}
```

The doPost() Method

- A POST request results from an HTML form that specifically lists POST as the METHOD and it should be handled by doPost() method.

  Syntax:

```
public void doPost (HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
        // Servlet code
}
```

The destroy() Method:

- The destroy() method is called only once at the end of the life cycle of a servlet.
- This method gives your servlet a chance to close database connections, halt background threads, write cookie lists or hit counts to disk, and perform other such cleanup activities.
- After the destroy() method is called, the servlet object is marked for garbage collection.

```
public void destroy()
{
// Finalization code...
}
```

2. **Write a java Servlet program to implement dynamic HTML using Servlet(Username and password should be accepted using HTML and displayed using servlet)**

```
package j2ee.prg1;
import  java.io.*;
import  javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import  javax.servlet.http.HttpServletRequest;
import  javax.servlet.http.HttpServletResponse;
public class program1 extends HttpServlet {
private static final long serialVersionUID = 1L;
public program1() {
super();
// TODO Auto-generated constructorstub
}
protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws
ServletException,
IOException
{
```

```
}
// TODOAuto-generated method stub
// Setting the HTTP Content-Type response headerto text/html
resp.setContentType("text/html");
// Returns a PrintWriter object that can send character text
//to the client.
PrintWriter pw=resp.getWriter();
//To retrieve the input values (name) from HTML page and store in the
string nameString name=req.getParameter("name");
// To retrieve the input values (pass) from HTMLpage and store in the
string passString pass=req.getParameter("pass");
//writing the output in the html
format
pw.println("<html><body>");
pw.println("Name is "+name+"<br>");
pw.println("Password is"+pass+"<br>");
pw.println("</body></html>");
}
Index.html
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<!-- send the form data to url mapping “pro1“ and the post method is used -->
<form method="post" action="pro1">
User name: <input type="text"
name="name"><br> Password: <input
type="password" name="pass"><br>
<input type="Submit" value="Accept">
</form>
</body>
</html>
```

3. Describe the classes and interfaces of javax.servlet package.

Two packages contain the classes and interfaces that are required to build servlets. These are javax.servlet and javax.servlet.http

Interface	Description
Servlet	Declares life cycle methods for a servlet.
ServletConfig	Allows servlets to get initialization parameters.
ServletContext	Enables servlets to log events and access information about their environment.
ServletRequest	Used to read data from a client request.
ServletResponse	Used to write data to a client response.

Class	Description
GenericServlet	Implements the **Servlet** and **ServletConfig** interfaces.
ServletInputStream	Provides an input stream for reading requests from a client.
ServletOutputStream	Provides an output stream for writing responses to a client.
ServletException	Indicates a servlet error occurred.
UnavailableException	Indicates a servlet is unavailable.

Method	Description
void destroy( )	Called when the servlet is unloaded.
ServletConfig getServletConfig( )	Returns a **ServletConfig** object that contains any initialization parameters.
String getServletInfo( )	Returns a string describing the servlet.
void init(ServletConfig *sc*) throws ServletException	Called when the servlet is initialized. Initialization parameters for the servlet can be obtained from *sc*. An **UnavailableException** should be thrown if the servlet cannot be initialized.
void service(ServletRequest *req*, ServletResponse *res*) throws ServletException, IOException	Called to process a request from a client. The request from the client can be read from *req*. The response to the client can be written to *res*. An exception is generated if a servlet or IO problem occurs.

TABLE 31-1 The Methods Defined by **Servlet**

The ServletConfig Interface

The **ServletConfig** interface allows a servlet to obtain configuration data when it is loaded. The methods declared by this interface are summarized here:

Method	Description
ServletContext getServletContext( )	Returns the context for this servlet.
String getInitParameter(String *param*)	Returns the value of the initialization parameter named *param*.
Enumeration getInitParameterNames( )	Returns an enumeration of all initialization parameter names.
String getServletName( )	Returns the name of the invoking servlet.

The Servlet Context interface

The ServletContext interface enables servlets to obtain information about their environment.

Method	Description
Object getAttribute(String *attr*)	Returns the value of the server attribute named *attr*.
String getMimeType(String *file*)	Returns the MIME type of *file*.
String getRealPath(String *vpath*)	Returns the real path that corresponds to the virtual path *vpath*.
String getServerInfo( )	Returns information about the server.
void log(String *s*)	Writes *s* to the servlet log.
void log(String *s*, Throwable *e*)	Writes *s* and the stack trace for *e* to the servlet log.
void setAttribute(String *attr*, Object *val*)	Sets the attribute specified by *attr* to the value passed in *val*.

TABLE 31-2 Various Methods Defined by **ServletContext**

ServletRequest Interface

Method	Description
Object getAttribute(String *attr*)	Returns the value of the attribute named *attr*.
String getCharacterEncoding( )	Returns the character encoding of the request.
int getContentLength( )	Returns the size of the request. The value –1 is returned if the size is unavailable.
String getContentType( )	Returns the type of the request. A **null** value is returned if the type cannot be determined.
ServletInputStream getInputStream( ) throws IOException	Returns a **ServletInputStream** that can be used to read binary data from the request. An **IllegalStateException** is thrown if **getReader( )** has already been invoked for this request.
String getParameter(String *pname*)	Returns the value of the parameter named *pname*.
Enumeration getParameterNames( )	Returns an enumeration of the parameter names for this request.
String[ ] getParameterValues(String *name*)	Returns an array containing values associated with the parameter specified by *name*.
String getProtocol( )	Returns a description of the protocol.
BufferedReader getReader( ) throws IOException	Returns a buffered reader that can be used to read text from the request. An **IllegalStateException** is thrown if **getInputStream( )** has already been invoked for this request.
String getRemoteAddr( )	Returns the string equivalent of the client IP address.
String getRemoteHost( )	Returns the string equivalent of the client host name.
String getScheme( )	Returns the transmission scheme of the URL used for the request (for example, “http”, “ftp”).
String getServerName( )	Returns the name of the server.
int getServerPort( )	Returns the port number.

TABLE 31-3 Various Methods Defined by **ServletRequest**

ServletResponse

Method	Description
String getCharacterEncoding( )	Returns the character encoding for the response.
ServletOutputStream getOutputStream( ) throws IOException	Returns a **ServletOutputStream** that can be used to write binary data to the response. An **IllegalStateException** is thrown if **getWriter( )** has already been invoked for this request.
PrintWriter getWriter( ) throws IOException	Returns a **PrintWriter** that can be used to write character data to the response. An **IllegalStateException** is thrown if **getOutputStream( )** has already been invoked for this request.
void setContentLength(int *size*)	Sets the content length for the response to *size*.
void setContentType(String *type*)	Sets the content type for the response to *type*.

TABLE 31-4 Various Methods Defined by **ServletResponse**

The GenericServlet Class:
The GenericServlet class provides implementations of the basic life cycle methods for a servlet. GenericServlet implements the Servlet and ServletConfig interfaces. In addition, a method to append a string to the server log file is available.
The signatures of this method are shown here:
void log(String s)
void log(String s, Throwable e)
Here, s is the string to be appended to the log, and e is an exception that occurred.
The ServletInputStream Class
The ServletInputStream class extends InputStream.
It is implemented by the servlet container and provides an input stream that a servlet developer can use to read the data from a client request.
It defines the default constructor.
A method is provided to read bytes from the stream.
intreadLine(byte[ ] buffer, int offset, int size) throws IOException
The ServletOutputStream Class:
The ServletOutputStream class extends OutputStream.
It is implemented by the servlet container and provides an output stream that a servlet developer can use to write data to a client response.
A default constructor is defined.
It also defines the print( ) and println( ) methods, which output data to the stream.

The ServletException Classes:

javax.servlet defines two exceptions.

- The first is ServletException, which indicates that a servlet problem has occurred.
- The second is UnavailableException, which extends ServletException. It indicates that a servlet is unavailable.

Javax.servlet.http

Interface	Description
HttpServletRequest	Enables servlets to read data from an HTTP request.
HttpServletResponse	Enables servlets to write data to an HTTP response.
HttpSession	Allows session data to be read and written.
HttpSessionBindingListener	Informs an object that it is bound to or unbound from a session.

4. Write a Servlet program to read data from a HTML form (gender data from radio buttons and colours data from check boxes) and display.

```
<html>
<head>
<title>TODO supply a title</title>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
</head>
<body>
<!-- send the form data to url mapping “prg3“ and the get method is used -->
<form method ="post" action="prg3">
<!--Display 3 Colors RED, BLUE, GREEN in the dropdown Box -->
<h1> Select your colors</h1>
<input type="checkbox" name="color" value="red"/>RED</br>
<input type="checkbox" name="color" value="green"/>GREEN</br>
<input type="checkbox" name="color" value="blue"/>BLUE</br>
<h1> Select your Course</h1>
UG:<input type="radio" name="course" value="ug"/><br>
PG:<input type="radio" name="course" value="pg"/><br>
<input type="submit" value="Submit"/>
</form>
</body>
</html>
```

Prg3.java

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class prg3 extends HttpServlet {
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
// Setting the HTTP Content-Type response header to text/html
response.setContentType("text/html");
// Returns a PrintWriter object out that can send character text to the client.
PrintWriter out=response.getWriter();
// To retrieve the optional values (color) from HTML page and store in the string color
String[] col = request.getParameterValues("color");
String cor = request.getParameter("course");
out.println("<html><body>");
out.println("Selected Colours");
for(String c:col)
out.println(c);
out.println("<br/>You have selected course "+cor);
```

```
out.println("</body></html>");
out.close();
}
}
```

5. Explain how do you handle HTTP requests and generate responses.

The HttpServlet class provides specialized methods that handle the various types of HTTP requests. A servlet developer typically overrides one of these methods. These methods are doDelete( ), doGet( ), doHead( ), doOptions( ), doPost( ), doPut( ), and doTrace( ). Acomplete description of the different types of HTTP requests is beyond the scope of this book. However, the GET and POST requests are commonly used when handling form input. Therefore, this section presents examples of these cases.

Handling HTTP GET Requests

The servlet is invoked when a form on a web page is submitted. The example contains two files. A web page is defined in ColorGet.htm, and a servlet is defined in ColorGetServlet.java. The HTML source code for ColorGet.htm is shown in the following listing. It defines a form that contains a select element and a submit button. Notice that the action parameter of the form tag specifies a URL.

The URLidentifies a servlet to process the HTTP GET request.

```
<html>
<body>
<center>
<form name="Form1"
 action="http://localhost:8080/servlets-examples/servlet/ColorGetServlet">
<B>Color:</B>
<select name="color" size="1">
<option value=”red”>RED</option>
<option value=”blue”>BLUE</option>
<option value=”green”>GREEN</option>
</select>
<br><br>
<input type=submit value="Submit">
</form>
</body>
</html>
```

The source code for ColorGetServlet.java is shown in the following listing. The doGet( ) method is overridden to process any HTTP GET requests that are sent to this servlet. It uses the getParameter( ) method of HttpServletRequest to obtain the selection that was made by the user. A response is then formulated.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class ColorGetServlet extends HttpServlet {
public void doGet(HttpServletRequest request,
HttpServletResponse response)
throws ServletException, IOException {
String color = request.getParameter("color");
```

```
response.setContentType("text/html");
PrintWriterpw = response.getWriter();
pw.println("<B>The selected color is:  ");
pw.println(color);
pw.close();
 }
 }
```

Handling Post requests:

The servlet is invoked when a form on a web page is submitted. The example contains two files. A web page is defined in ColorPost.htm, and a servlet is defined in ColorPostServlet.java.

The HTMLsourcecode for ColorPost.htm is shown in the following listing. It is identical to ColorGet.htm except that the method parameter for the form tag explicitly specifies that the POST method should be used, and the action parameter for the form tag specifies a different servlet.

```
<html>
<body>
<center>
<form name="Form1"
 method="post"
 action="http://localhost:8080/servlets-examples/servlet/ColorPostServlet">
<B>Color:</B>
<select name="color" size="1">
<option value="Red">Red</option>
<option value="Green">Green</option>
<option value="Blue">Blue</option>
</select>
<br><br>
<input type=submit value="Submit">
</form>
</body>
</html>
```

The source code for ColorPostServlet.java is shown in the following listing. The doPost( ) method is overridden to process any HTTP POST requests that are sent to this servlet. It uses the getParameter( ) method of HttpServletRequest to obtain the selection that was made by the user. A response is then formulated.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class ColorPostServlet extends HttpServlet {
public void doPost(HttpServletRequest request,
HttpServletResponse response)
throws ServletException, IOException {
String color = request.getParameter("color");
response.setContentType("text/html");
PrintWriterpw = response.getWriter();
pw.println("<B>The selected color is:  ");
```

```
pw.println(color);
pw.close();
 }
 }
```

6. Write a java program to create a cookie to store user preferences

```
<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title>Select Your Preferred Color</title>

</head>

<body>

<h2>Choose a Color:</h2>

<form action="store" method="post">

      RED: <input type="radio" name="color" value="RED"/><br>

      GREEN: <input type="radio" name="color" value="GREEN"/><br>

      BLUE: <input type="radio" name="color" value="BLUE"/><br>

<input type="submit" value="Submit"/>

</form>

</body>

</html>
```

Store.java

```
package j2ee.prg4;


import java.io.IOException;

import java.io.PrintWriter;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;
```

```
import javax.servlet.http.Cookie;

@WebServlet("/store")
public class Store extends HttpServlet {
   private static final long serialVersionUID = 1L;

   public Store() {
super();
   }

   protected void doPost(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {
response.setContentType("text/html;charset=UTF-8");
PrintWriter out = response.getWriter();

      try {
         // Get selected color from request
         String color = request.getParameter("color");

         if (color != null && (color.equals("RED") || color.equals("BLUE") || color.equals("GREEN"))) {
            // Create a cookie to store color preference
            Cookie colorCookie = new Cookie("color", color);
colorCookie.setMaxAge(60 * 60 * 24 * 7); // Cookie expires in 7 days
response.addCookie(colorCookie);

            // Display confirmation message
out.println("<html><body>");
out.println("You selected: " + color + "<br>");
out.println("<form action='retrieve' method='post'>");
```

```
out.println("<input type='submit' value='View Stored Preference'/>");
out.println("</form>");
out.println("</body></html>");
        } else {
out.println("Invalid selection. Please go back and choose a color.");
        }
      } finally {
out.close();
      }
   }
}
```

Retrieve.java

```
package j2ee.prg4;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.Cookie;

@WebServlet("/retrieve")
public class Retrieve extends HttpServlet {
   private static final long serialVersionUID = 1L;

   public Retrieve() {
super();
```

```
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
response.setContentType("text/html;charset=UTF-8");
PrintWriter out = response.getWriter();

        try {
            // Get cookies from request
Cookie[] cookies = request.getCookies();
            String color = "WHITE"; // Default background color

            if (cookies != null) {
                for (Cookie cookie : cookies) {
                    if (cookie.getName().equals("color")) {
color = cookie.getValue();
                        break;
                    }
                }
            }

            // Apply the color and display it
out.println("<html><head><title>Stored Preference</title></head>");
out.println("<body bgcolor='" + color + "'>");
out.println("<h2>Your Selected Color: " + color + "</h2>");
out.println("<a href='index.html'>Change Color</a>");
out.println("</body></html>");
        } finally {
out.close();
```

```
    }
  }
}
```

7. **What is JSP. Explain different types of JSP tags by taking suitable examples.**

JSP scriptlet tag: A scriptlet tag java source code in JSP.

<% java source code %>

In this example, we are displaying a welcome message.

```
<html>
<body>
<% out.print(“Welcome to jsp” %>
</body>
</html>
```

JSP Declaration Tag: The JSP declaration tag is used to declare variables, objects and methods. The code written inside the jsp declaration tag is placed outside the service() method of auto generated servlet. So it doesn't get memory at each request.

<%! Field or method declaration %>

Ex:

```
<html>
<body>
<%! Int data=50; %>
<%= “value “ +data %>
</body>
</html>
```

JSP Expression Tag:

Expression Tag is used to print out java language expression that is put between the tags. An expression tag can hold any java language expression that can be used as an argument to the out.print() method.

Syntax :<%= java expression %>

<%= (2*5) %>

JSP Directives:

The jsp directives are messages that tells the web container how to translate a JSP page into the corresponding servlet.

Syntax

<%@ directive attribute=”value” %>

There are three types of directives: 1. import 2. include directive directive 3. taglib directive

8. **Write a JAVA Servlet Program to Auto Web Page Refresh (Consider a webpage which is displaying Date and time)**

```
@WebServlet("/progra m2")
public class program2 extends HttpServlet {
```

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException
{
response.setContentType("text/html");
response.addHeader("Refresh","1");
PrintWriter out=response.getWriter();
out.println("Text servlet says hi at "+new Date());
}
}
```

9. **List and explain any 5 collection algorithms. Demonstrate various algorithms with an example program.**

Method	Description
static <T> boolean addAll(Collection <? super T> *c*, T ... *elements*)	Inserts the elements specified by *elements* into the collection specified by *c*. Returns **true** if the elements were added and **false** otherwise.
static <T> Queue<T> asLifoQueue(Deque<T> *c*)	Returns a last-in, first-out view of *c*. (Added by Java SE 6.)
static <T> int binarySearch(List<? extends T> *list*, T *value*, Comparator<? super T> *c*)	Searches for *value* in *list* ordered according to *c*. Returns the position of *value* in *list*, or a negative value if *value* is not found.
static <T> int binarySearch(List<? extends Comparable<? super T>> *list*, T *value*)	Searches for *value* in *list*. The list must be sorted. Returns the position of *value* in *list*, or a negative value if *value* is not found.
static <E> Collection<E> checkedCollection(Collection<E> *c*, Class<E> *t*)	Returns a run-time type-safe view of a collection. An attempt to insert an incompatible element will cause a **ClassCastException**.
static <E> List<E> checkedList(List<E> *c*, Class<E> *t*)	Returns a run-time type-safe view of a **List**. An attempt to insert an incompatible element will cause a **ClassCastException**.
static <K, V> Map<K, V> checkedMap(Map<K, V> *c*, Class<K> *keyT*, Class<V> *valueT*)	Returns a run-time type-safe view of a **Map**. An attempt to insert an incompatible element will cause a **ClassCastException**.
static <E> List<E> checkedSet(Set<E> *c*, Class<E> *t*)	Returns a run-time type-safe view of a **Set**. An attempt to insert an incompatible element will cause a **ClassCastException**.
static <K, V> SortedMap<K, V> checkedSortedMap(SortedMap<K, V> *c*, Class<K> *keyT*, Class<V> *valueT*)	Returns a run-time type-safe view of a **SortedMap**. An attempt to insert an incompatible element will cause a **ClassCastException**.

static <T> void copy(List<? super T> *list1*, List<? extends T> *list2*)	Copies the elements of *list2* to *list1*.
static boolean disjoint(Collection<?> *a*, Collection<?> *b*)	Compares the elements in *a* to elements in *b*. Returns **true** if the two collections contain no common elements (i.e., the collections contain disjoint sets of elements). Otherwise, returns **true**.
static <T> List<T> emptyList( )	Returns an immutable, empty **List** object of the inferred type.
static <K, V> Map<K, V> emptyMap( )	Returns an immutable, empty **Map** object of the inferred type.
static <T> Set<T> emptySet( )	Returns an immutable, empty **Set** object of the inferred type.
static <T> Enumeration<T> enumeration(Collection<T> *c*)	Returns an enumeration over *c*. (See "The Enumeration Interface," later in this chapter.)
static <T> void fill(List<? super T> *list*, T *obj*)	Assigns *obj* to each element of *list*.

TABLE 17-14 The Algorithms Defined by **Collections**

Example:

```
// Demonstrate various algorithms.
 import java.util.*;
 class AlgorithmsDemo {
 public static void main(String args[]) {
 // Create and initialize linked list.
LinkedList<Integer>ll = new LinkedList<Integer>();
ll.add(-8);
ll.add(20);
ll.add(-20);
ll.add(8);
 // Create a reverse order comparator.
 Comparator<Integer> r = Collections.reverseOrder();
 // Sort list by using the comparator.
Collections.sort(ll, r);
System.out.print("List sorted in reverse: ");
for(inti : ll)
System.out.print(i+ " ");
System.out.println();
 // Shuffle list.
Collections.shuffle(ll);
 // Display randomized list.
System.out.print("List shuffled: ");
for(inti : ll)
System.out.print(i + " ");
System.out.println();
System.out.println("Minimum: " + Collections.min(ll)); System.out.println("Maximum: " +
Collections.max(ll)); } }
```

10. Explain the need of generic collections

the Collections Framework is arguably the single most important use of generics in the Java API. The reason for this is that generics add type safety to the Collections Framework.
The following program stores
a list of strings in an ArrayList and then displays the contents of the list:

```
// Pre-generics example that uses a collection.
import java.util.*;
class OldStyle {
public static void main(String args[]) {
ArrayList list = new ArrayList();
// These lines store strings, but any type of object
// can be stored.  In old-style code, there is no
// convenient way to restrict the type of objects stored
// in a collection
list.add("one");
list.add("two");
list.add("three");
list.add("four");
Iterator itr = list.iterator();
while(itr.hasNext()) {
// To retrieve an element, an explicit type cast is needed
// because the collection stores only Object.
String str = (String) itr.next(); // explicit cast needed here.
System.out.println(str + " is " + str.length() + " chars long.");
}
}
}
```

Prior to generics, all collections stored references of type Object. This allowed any type of reference to be stored in the collection. The above program uses this feature to store references to objects of type String in list, but any type of reference could have been stored.
Unfortunately, the fact that a pre-generics collection stored Object references could easily lead to errors. First, it required that you, rather than the compiler, ensure that only objects of the proper type be stored in a specific collection. For example, in the preceding example, list is clearly intended to store Strings, but there is nothing that actually prevents another type of reference from being added to the collection.
The second problem with pre-generics collections is that when you retrieve a reference from the collection, you must manually cast that reference into the proper type. This is why the above program casts the reference returned by next( ) into String. Prior to generics, collections simply stored Object references. Thus, the cast was necessary when retrieving objects from a collection.
The addition of generics fundamentally improves the usability and safety of collections because it

- Ensures that only references to objects of the proper type can actually be stored in a collection. Thus, a collection will always contain references of a known type.
- Eliminates the need to cast a reference retrieved from a collection. Instead, a reference retrieved from a collection is automatically cast into the proper type. This prevents run-time errors due to invalid casts and avoids an entire category of errors.