Tên: Nguyễn Hải Đăng

Mã sinh viên:PC03033

Lớp:SD17301

Bài báo cáo lab4

Bài 1 (2 điểm)

-Hãy cài đặt mã nguồn cho các phương thức làm việc với parameter được đặc tả

trong lớp Spring Bean (ParamService)

+Thông tin code:

 /\*\*

     \* Đọc chuỗi giá trị của tham số

     \*

     \* @param name         tên tham số

     \* @param defaultValue giá trị mặc định

     \* @return giá trị tham số hoặc giá trị mặc định nếu không tồn tại

     \*/

    public String getString(String name, String defaultValue) {

        String value = request.getParameter(name);

        return value != null ? value : defaultValue;

    }

1. public String getString(String name, String defaultValue): Đây là khai báo của phương thức có tên getString với hai tham số đầu vào là name và defaultValue. Phương thức trả về một chuỗi (String).
2. String value = request.getParameter(name);: Dòng này lấy giá trị của tham số name từ một đối tượng request. Đối tượng request có thể là một đối tượng HttpServletRequest trong Java Servlet hoặc một đối tượng tương tự trong môi trường khác. Phương thức getParameter(name) trả về giá trị của tham số với tên tương ứng (name).
3. return value != null ? value : defaultValue;: Dòng này sử dụng toán tử ba ngôi để kiểm tra giá trị của biến value. Nếu value khác null, tức là có giá trị được truyền vào tham số name, thì phương thức trả về giá trị đó (value). Nếu value là null, tức là không có giá trị được truyền vào tham số name, thì phương thức trả về giá trị mặc định (defaultValue).

 /\*\*

     \* Đọc số nguyên giá trị của tham số

     \*

     \* @param name         tên tham số

     \* @param defaultValue giá trị mặc định

     \* @return giá trị tham số hoặc giá trị mặc định nếu không tồn tại

     \*/

    public int getInt(String name, int defaultValue) {

        String value = request.getParameter(name);

        try {

            return value != null ? Integer.parseInt(value) : defaultValue;

        } catch (NumberFormatException e) {

            return defaultValue;

        }

    }

1. public int getInt(String name, int defaultValue): Đây là khai báo của phương thức có tên getInt với hai tham số đầu vào là name và defaultValue. Phương thức trả về một số nguyên (int).
2. String value = request.getParameter(name);: Dòng này lấy giá trị của tham số name từ một đối tượng request, tương tự như trong đoạn mã trước. Đối tượng request có thể là một đối tượng HttpServletRequest trong Java Servlet hoặc một đối tượng tương tự trong môi trường khác.
3. try { return value != null ? Integer.parseInt(value) : defaultValue; } catch (NumberFormatException e) { return defaultValue; }: Đoạn mã này sử dụng một khối try-catch để xử lý việc chuyển đổi giá trị từ chuỗi (String) sang số nguyên (int).
   1. Dòng return value != null ? Integer.parseInt(value) : defaultValue; kiểm tra giá trị của biến value. Nếu value khác null, tức là có giá trị được truyền vào tham số name, thì phương thức sử dụng Integer.parseInt(value) để chuyển đổi giá trị từ chuỗi sang số nguyên và trả về giá trị đó. Nếu value là null, tức là không có giá trị được truyền vào tham số name, phương thức trả về giá trị mặc định (defaultValue).
   2. Dòng catch (NumberFormatException e) { return defaultValue; } là khối catch, nếu trong quá trình chuyển đổi chuỗi sang số nguyên xảy ra lỗi NumberFormatException, nghĩa là chuỗi không thể chuyển thành số nguyên, thì phương thức trả về giá trị mặc định (defaultValue).

/\*\*

     \* Đọc số thực giá trị của tham số

     \*

     \* @param name         tên tham số

     \* @param defaultValue giá trị mặc định

     \* @return giá trị tham số hoặc giá trị mặc định nếu không tồn tại

     \*/

    public double getDouble(String name, double defaultValue) {

        String value = request.getParameter(name);

        try {

            return value != null ? Double.parseDouble(value) : defaultValue;

        } catch (NumberFormatException e) {

            return defaultValue;

        }

    }

* 1. public double getDouble(String name, double defaultValue): Đây là khai báo của phương thức có tên getDouble với hai tham số đầu vào là name và defaultValue. Phương thức trả về một số thực (double).
  2. String value = request.getParameter(name);: Dòng này lấy giá trị của tham số name từ một đối tượng request, tương tự như trong đoạn mã trước.
  3. try { return value != null ? Double.parseDouble(value) : defaultValue; } catch (NumberFormatException e) { return defaultValue; }: Đoạn mã này sử dụng một khối try-catch để xử lý việc chuyển đổi giá trị từ chuỗi (String) sang số thực (double).
     1. Dòng return value != null ? Double.parseDouble(value) : defaultValue; kiểm tra giá trị của biến value. Nếu value khác null, tức là có giá trị được truyền vào tham số name, phương thức sử dụng Double.parseDouble(value) để chuyển đổi giá trị từ chuỗi sang số thực và trả về giá trị đó. Nếu value là null, tức là không có giá trị được truyền vào tham số name, phương thức trả về giá trị mặc định (defaultValue).
     2. Dòng catch (NumberFormatException e) { return defaultValue; } là khối catch, nếu trong quá trình chuyển đổi chuỗi sang số thực xảy ra lỗi NumberFormatException, nghĩa là chuỗi không thể chuyển thành số thực, thì phương thức trả về giá trị mặc định (defaultValue).

/\*\*

     \* Đọc giá trị boolean của tham số

     \*

     \* @param name         tên tham số

     \* @param defaultValue giá trị mặc định

     \* @return giá trị tham số hoặc giá trị mặc định nếu không tồn tại

     \*/

    public boolean getBoolean(String name, boolean defaultValue) {

        String value = request.getParameter(name);

        return value != null ? Boolean.parseBoolean(value) : defaultValue;

    }

* + 1. public boolean getBoolean(String name, boolean defaultValue): Đây là khai báo của phương thức có tên getBoolean với hai tham số đầu vào là name và defaultValue. Phương thức trả về một giá trị boolean.
    2. String value = request.getParameter(name);: Dòng này lấy giá trị của tham số name từ một đối tượng request, tương tự như trong các đoạn mã trước.
    3. return value != null ? Boolean.parseBoolean(value) : defaultValue;: Dòng này sử dụng toán tử ba ngôi để kiểm tra giá trị của biến value. Nếu value khác null, tức là có giá trị được truyền vào tham số name, phương thức sử dụng Boolean.parseBoolean(value) để chuyển đổi giá trị từ chuỗi sang giá trị boolean và trả về giá trị đó. Nếu value là null, tức là không có giá trị được truyền vào tham số name, phương thức trả về giá trị mặc định (defaultValue).

/\*\*

     \* Đọc giá trị thời gian của tham số

     \*

     \* @param name    tên tham số

     \* @param pattern là định dạng thời gian

     \* @return giá trị tham số hoặc null nếu không tồn tại

     \* @throws RuntimeException lỗi sai định dạng

     \*/

    public Date getDate(String name, String pattern) {

        String value = request.getParameter(name);

        SimpleDateFormat dateFormat = new SimpleDateFormat(pattern);

        try {

            return value != null ? dateFormat.parse(value) : null;

        } catch (ParseException e) {

            throw new RuntimeException("Lỗi sai định dạng thời gian: " + e.getMessage(), e);

        }

    }

* + 1. public Date getDate(String name, String pattern): Đây là khai báo của phương thức có tên getDate với hai tham số đầu vào là name và pattern. Phương thức trả về một đối tượng Date hoặc null nếu không tồn tại giá trị.
    2. String value = request.getParameter(name);: Dòng này lấy giá trị của tham số name từ đối tượng request, tương tự như các đoạn mã trước.
    3. SimpleDateFormat dateFormat = new SimpleDateFormat(pattern);: Đoạn này khởi tạo một đối tượng SimpleDateFormat để định dạng thời gian dựa trên mẫu (pattern) được truyền vào. Đối tượng này sẽ được sử dụng để chuyển đổi chuỗi thời gian thành đối tượng Date.
    4. try { return value != null ? dateFormat.parse(value) : null; } catch (ParseException e) { throw new RuntimeException("Lỗi sai định dạng thời gian: " + e.getMessage(), e); }: Đoạn này sử dụng khối try-catch để xử lý chuyển đổi chuỗi thời gian thành đối tượng Date.
       1. Dòng return value != null ? dateFormat.parse(value) : null; kiểm tra giá trị của biến value. Nếu value khác null, tức là có giá trị được truyền vào tham số name, phương thức sử dụng dateFormat.parse(value) để chuyển đổi chuỗi thời gian thành đối tượng Date và trả về đối tượng Date đó. Nếu value là null, tức là không có giá trị được truyền vào tham số name, phương thức trả về null.
       2. Dòng catch (ParseException e) { throw new RuntimeException("Lỗi sai định dạng thời gian: " + e.getMessage(), e); } là khối catch, nếu trong quá trình chuyển đổi chuỗi thời gian xảy ra lỗi ParseException, nghĩa là chuỗi không thể chuyển thành đối tượng Date theo định dạng đã cho, phương thức sẽ ném một ngoại lệ RuntimeException với thông báo lỗi tương ứng.

/\*\*

     \* Lưu file upload vào thư mục

     \*

     \* @param file chứa file upload từ client

     \* @param path đường dẫn tính từ webroot

     \* @return đối tượng chứa file đã lưu hoặc null nếu không có file upload

     \* @throws RuntimeException lỗi lưu file

     \*/

    public File save(MultipartFile file, String path) {

        if (file != null && !file.isEmpty()) {

            try {

                File directory = new File(path);

                if (!directory.exists()) {

                    directory.mkdirs();

                }

                String filename = file.getOriginalFilename();

                File saveFile = new File(directory, filename);

                Files.copy(file.getInputStream(), saveFile.toPath(), StandardCopyOption.REPLACE\_EXISTING);

                System.out.println(saveFile.getAbsolutePath());

                return saveFile;

            } catch (IOException e) {

                throw new RuntimeException("Lỗi lưu file: " + e.getMessage(), e);

            }

        }

        return null;

    }

* + - 1. public File save(MultipartFile file, String path): Đây là khai báo của phương thức có tên save với hai tham số đầu vào là file (đối tượng chứa file upload từ client) và path (đường dẫn tới thư mục lưu trữ file tính từ webroot). Phương thức trả về một đối tượng File chứa file đã lưu hoặc null nếu không có file upload.
      2. if (file != null && !file.isEmpty()) {: Dòng này kiểm tra xem file có tồn tại và không rỗng. Nếu file tồn tại và không rỗng, tiếp tục thực hiện lưu file. Nếu không, phương thức trả về null.
      3. File directory = new File(path);: Dòng này tạo một đối tượng File từ path, đại diện cho thư mục lưu trữ file.
      4. if (!directory.exists()) { directory.mkdirs(); }: Dòng này kiểm tra xem thư mục lưu trữ file đã tồn tại hay chưa. Nếu chưa tồn tại, phương thức tạo thư mục bằng cách gọi phương thức mkdirs().
      5. String filename = file.getOriginalFilename();: Dòng này lấy tên gốc của file từ đối tượng file (đối tượng MultipartFile) bằng phương thức getOriginalFilename().
      6. File saveFile = new File(directory, filename);: Dòng này tạo một đối tượng File mới để lưu file trong thư mục đã chỉ định, sử dụng directory và filename.
      7. Files.copy(file.getInputStream(), saveFile.toPath(), StandardCopyOption.REPLACE\_EXISTING);: Dòng này thực hiện sao chép nội dung của file được upload từ đối tượng file vào đường dẫn saveFile. Để sao chép file, phương thức sử dụng phương thức copy từ lớp Files trong gói java.nio.file. Tùy chọn StandardCopyOption.REPLACE\_EXISTING được sử dụng để ghi đè file nếu file đã tồn tại.
      8. System.out.println(saveFile.getAbsolutePath());: Dòng này in ra đường dẫn tuyệt đối của file đã được lưu trong console (có thể để mục đích debug).
      9. return saveFile;: Dòng này trả về đối tượng File chứa file đã lưu.
      10. catch (IOException e) { throw new RuntimeException("Lỗi lưu file: " + e.getMessage(), e); }: Đoạn này sử dụng khối try-catch để xử lý các lỗi xảy ra trong quá trình lưu file. Nếu xảy ra lỗi IOException (lỗi I/O), phương thức ném một ngoại lệ RuntimeException với thông báo lỗi tương ứng.

-CookieService: Hãy cài đặt mã nguồn cho các phương thức làm việc với cookie được

đặc tả trong lớp Spring Bean (CookieService)

+Thông tin code:

@Autowired

    private HttpServletRequest request;

    @Autowired

    private HttpServletResponse response;

/\*\*

     \* Đọc cookie từ request

     \*

     \* @param name tên cookie cần đọc

     \* @return đối tượng cookie đọc được hoặc null nếu không tồn tại

     \*/

    public Cookie get(String name) {

        Cookie[] cookies = request.getCookies();

        if (cookies != null) {

            for (Cookie cookie : cookies) {

                if (cookie.getName().equals(name)) {

                    return cookie;

                }

            }

        }

        return null;

    }

1. public Cookie get(String name): Đây là khai báo của phương thức có tên get với một tham số đầu vào là name (tên cookie cần đọc). Phương thức trả về một đối tượng Cookie đọc được hoặc null nếu không tồn tại.
2. Cookie[] cookies = request.getCookies();: Dòng này lấy danh sách các cookie từ đối tượng request bằng cách sử dụng phương thức getCookies(). Đối tượng request được tiêm vào (inject) thông qua chú thích @Autowired.
3. if (cookies != null) {: Dòng này kiểm tra xem danh sách cookie có tồn tại hay không. Nếu danh sách không null, tiếp tục thực hiện việc tìm kiếm cookie.
4. for (Cookie cookie : cookies) {: Dòng này bắt đầu một vòng lặp for-each để duyệt qua từng cookie trong danh sách.
5. if (cookie.getName().equals(name)) {: Dòng này kiểm tra xem tên của cookie hiện tại có trùng khớp với name được truyền vào hay không.
6. return cookie;: Dòng này trả về cookie hiện tại nếu tên của cookie trùng khớp với name.
7. return null;: Dòng này trả về null nếu không tìm thấy cookie có tên trùng khớp trong danh sách.
8. /\*\*
9. \* Đọc giá trị của cookie từ request
10. \*
11. \* @param name tên cookie cần đọc
12. \* @return chuỗi giá trị đọc được hoặc rỗng nếu không tồn tại
13. \*/
14. public String getValue(String name) {
15. Cookie cookie = get(name);
16. return cookie != null ? cookie.getValue() : "";
17. }
18. public String getValue(String name): Đây là khai báo của phương thức có tên getValue với một tham số đầu vào là name (tên cookie cần đọc). Phương thức trả về một chuỗi giá trị đọc được hoặc chuỗi rỗng nếu không tồn tại.
19. Cookie cookie = get(name);: Dòng này gọi phương thức get(name) để lấy đối tượng Cookie có tên trùng khớp với name.
20. return cookie != null ? cookie.getValue() : "";: Dòng này trả về giá trị của cookie nếu cookie khác null, ngược lại trả về chuỗi rỗng.

/\*\*

     \* Tạo và gửi cookie về client

     \*

     \* @param name  tên cookie

     \* @param value giá trị cookie

     \* @param hours thời hạn (giờ)

     \* @return đối tượng cookie đã tạo

     \*/

    public Cookie add(String name, String value, int hours) {

        Cookie cookie = new Cookie(name, value);

        cookie.setMaxAge(hours \* 60 \* 60);

        cookie.setPath("/");

        response.addCookie(cookie);

        return cookie;

    }

1. public Cookie add(String name, String value, int hours): Đây là khai báo của phương thức có tên add với ba tham số đầu vào: name (tên cookie), value (giá trị cookie), và hours (thời hạn của cookie tính bằng giờ). Phương thức trả về một đối tượng Cookie đã tạo.
2. Cookie cookie = new Cookie(name, value);: Dòng này tạo một đối tượng Cookie mới với tên là name và giá trị là value.
3. cookie.setMaxAge(hours \* 60 \* 60);: Dòng này đặt thời hạn của cookie bằng cách gán số giờ được truyền vào (hours) nhân với 60 (phút) nhân với 60 (giây) bằng tổng số giây. Phương thức setMaxAge() được sử dụng để đặt thời hạn sống của cookie trong số giây.
4. cookie.setPath("/");: Dòng này đặt đường dẫn cho cookie bằng cách gán "/" vào phương thức setPath(). Điều này chỉ định rằng cookie có thể được sử dụng trên tất cả các đường dẫn trong domain hiện tại.
5. response.addCookie(cookie);: Dòng này thêm cookie vào đối tượng response (phản hồi) để gửi về cho client.
6. return cookie;: Dòng này trả về đối tượng Cookie đã tạo.

/\*\*

     \* Xóa cookie khỏi client

     \*

     \* @param name tên cookie cần xóa

     \*/

    public void remove(String name) {

        Cookie cookie = get(name);

        if (cookie != null) {

            cookie.setMaxAge(0);

            cookie.setPath("/");

            response.addCookie(cookie);

        }

    }

1. public void remove(String name): Đây là khai báo của phương thức có tên remove với một tham số đầu vào là name (tên cookie cần xóa). Phương thức không trả về giá trị (void).
2. Cookie cookie = get(name);: Dòng này gọi phương thức get(name) để lấy đối tượng Cookie có tên trùng khớp với name.
3. if (cookie != null) {: Dòng này kiểm tra xem cookie có tồn tại hay không. Nếu cookie khác null, tiếp tục thực hiện việc xóa cookie.
4. cookie.setMaxAge(0);: Dòng này đặt thời hạn của cookie bằng 0. Điều này khiến cookie bị hết hạn ngay lập tức.
5. cookie.setPath("/");: Dòng này đặt đường dẫn cho cookie bằng cách gán "/" vào phương thức setPath(). Điều này đảm bảo cookie sẽ được xóa trên tất cả các đường dẫn trong domain hiện tại.
6. response.addCookie(cookie);: Dòng này thêm cookie vào đối tượng response (phản hồi) để gửi yêu cầu xóa cookie về cho client.

- SessionService: Hãy cài đặt mã nguồn cho các phương thức làm việc với attribute

trong session được đặc tả trong lớp Spring Bean (SessionService)

+Thông tin code:

/\*\*

    \* Đọc giá trị của attribute trong session

    \* @param name tên attribute

    \* @return giá trị đọc được hoặc null nếu không tồn tại

    \*/

    public <T> T get(String name) {

        return (T) session.getAttribute(name);

    }

1. public <T> T get(String name): Đây là khai báo của phương thức có tên get với một tham số đầu vào là name (tên thuộc tính). Phương thức trả về một giá trị có kiểu T, và kiểu T là kiểu dữ liệu được xác định khi gọi phương thức.
2. return (T) session.getAttribute(name);: Dòng này sử dụng phương thức getAttribute(name) của đối tượng session để lấy giá trị của thuộc tính có tên trùng khớp với name. Vì phương thức getAttribute() trả về một đối tượng kiểu Object, nên để trả về kiểu dữ liệu T, ta ép kiểu (cast) giá trị trước khi trả về.

/\*\*

    \* Thay đổi hoặc tạo mới attribute trong session

    \* @param name tên attribute

    \* @param value giá trị attribute

    \*/

    public void set(String name, Object value) {

        session.setAttribute(name, value);

    }

1. public void set(String name, Object value): Đây là khai báo của phương thức có tên set với hai tham số đầu vào là name (tên thuộc tính) và value (giá trị thuộc tính). Phương thức không trả về giá trị (void).
2. session.setAttribute(name, value);: Dòng này sử dụng phương thức setAttribute(name, value) của đối tượng session để thay đổi hoặc tạo mới thuộc tính có tên là name với giá trị là value.

/\*\*

    \* Xóa attribute trong session

    \* @param name tên attribute cần xóa

    \*/

    public void remove(String name) {session.removeAttribute(name);}

1. public void remove(String name): Đây là khai báo của phương thức có tên remove với một tham số đầu vào là name (tên thuộc tính). Phương thức không trả về giá trị (void).
2. session.removeAttribute(name);: Dòng này sử dụng phương thức removeAttribute(name) của đối tượng session để xóa thuộc tính có tên là name khỏi session.

Bài 2 (2 điểm)

Tạo AccountController để xử lý đăng nhập có cấu trúc như sau

@Controller

    public class AccountController {

        @GetMapping("/account/login")

        public String login1() {

            return "/account/login";

        }

        @PostMapping("/account/login")

        public String login2() {

            return "/account/login";

        }

    }

Tạo view login.html có chứa form đăng nhập như sau

<form action="/account/login" method="post">

        <input type="text" name="username" placeholder="Username"> <input

            type="password" name="password" placeholder="Password"> <input

            type="checkbox" name="remember" value="true"> Remember me

        <button type="submit">Login</button>

    </form>

Tiêm ParamService, CookieService vào AccountController để làm việc với tham số

và cookie:

@Autowired

    private CookieService cookieService;

    @Autowired

    private ParamService paramService;

    @Autowired

    private SessionService sessionService;

Viết mã để đọc các tham số (username, password, remember) và ghi nhớ tài khoản

đăng nhập được cho login2():

String un = paramService.getString("username", "");

String pw = paramService.getString("password", "");

boolean rm = paramService.getBoolean("remember", false);

Nếu đăng nhập thành công (un=”poly”, pw=”123”)

if (un.equals("poly") && pw.equals("123")) {

            // Lưu username vào session

            sessionService.set("username", un);

            if (rm) {

                // Ghi nhớ tài khoản trong 10 ngày

                cookieService.add("user", un, 10);

            } else {

                // Xóa cookie tài khoản đã ghi nhớ trước đó

                cookieService.remove("user");

            }

            // Chuyển hướng đến trang thành công

            return "redirect:/account/success";

        }

        // Chuyển hướng đến trang đăng nhập với thông báo lỗi

        return "redirect:/account/login?error=true";

Bài 3 (1 điểm)

Giảng viên cho thêm đăng ký có upload hình và sử dụng ParamService.save() để lưu

hình.

-Phần đăng ký có upload file:

 @GetMapping("/account/register")

    public String register1() {

        return "/account/register";

    }

    @PostMapping("/account/register")

    public String register2(@RequestParam("file") MultipartFile file) {

        // Lưu hình vào thư mục

        String imagePath = "/path/to/image/folder";

        File savedImage = paramService.save(file, imagePath);

        // Xử lý đăng ký và lưu thông tin vào database

        System.out.println(savedImage);

        // Chuyển hướng đến trang thành công

        return "redirect:/account/success";

    }

Form register.html:

<form action="/account/register" method="post"

        enctype="multipart/form-data">

        <input type="text" name="username" placeholder="Username"> <input

            type="password" name="password" placeholder="Password"> <input

            type="file" name="file">

        <button type="submit">Register</button>

    </form>

PHẦNII

Bài 4 (2 điểm)

-Cho interface ShoppingCartService mô tả các hoạt động của giỏ hàng như sau:

public interface ShoppingCartService {

    Item add(Integer id);

    void remove(Integer id);

    Item update(Integer id, int qty);

    void clear();

    Collection<Item> getItems();

    int getCount();

    double getAmount();

}

- Trong đó lớp Item mô tả thông tin của mặt hàng như sau:

@Data

@NoArgsConstructor

@AllArgsConstructor

public class Item {

    private Integer id;

    private String name;

    private double price;

    private int qty = 1;

}

-Hãy tạo lớp Spring Bean ShoppingCartServiceImpl và cài đặt mã nguồn cho các

phương thức thực hiện như đã đặc tả trong interface ShoppingCartService.

Hướng dẫn:

-Sử dụng @Service và @SessionScope để khai báo cho ShoppingCartServiceImpl để

Spring Bean được tạo theo từng phiên làm việc.

-Có thể sử dụng List<Item> hoặc Map<Integer, Item> để chứa các mặt hàng đã chọn.

Nếu dùng map thì việc truy xuất và kiểm tra sự tồn tại của một mặt hàng trong giỏ

sẽ thuận tiện hơn.

+Thông tin code:

private Map<Integer, Item> map = new HashMap<>();

    ShoppingCartServiceImpl cartServiceImpl;

Đoạn mã khai báo biến map và cartServideImpl

@Override

    public Item add(Integer id) {

        Item item = map.get(id);

        if (item == null) {

            // Mặt hàng chưa tồn tại trong giỏ hàng

            // Thêm mặt hàng vào giỏ hàng với số lượng ban đầu là 1

            DB db = new DB(); // Tạo đối tượng DB

            Map<Integer, Item> dbItems = db.items; // Lấy danh sách mặt hàng từ DB

            Item dbItem = dbItems.get(id); // Lấy mặt hàng từ DB dựa trên ID

            if (dbItem != null) {

                // Copy thông tin từ DBItem sang item mới

                item = new Item(dbItem.getId(), dbItem.getName(), dbItem.getPrice(), 1);

                map.put(id, item);

            }

        } else {

            // Mặt hàng đã tồn tại trong giỏ hàng

            // Tăng số lượng lên 1

            item.setQty(item.getQty() + 1);

        }

        return item;

    }

Đoạn mã thêm dữ liệu:

Lưu ý dữ liệu được lấy từ class DB của bài 5

Sau khi lấy được dữ liệu sẽ tạo model item mới và đưa dữ liệu được lấy vào Item

Nếu đã trùng id thì sẽ cộng số lượng thêm 1

@Override

    public void remove(Integer id) {

        map.remove(id);

    }

Xóa Item khỏi giỏ hàng

@Override

    public Item update(Integer id, int qty) {

        Item item = map.get(id);

        if (item != null) {

            // Cập nhật số lượng của mặt hàng

            item.setQty(qty);

        }

        return item;

    }

Cập nhật số lượng của Item

@Override

    public void clear() {

        map.clear();

    }

Xóa hết toàn bộ Item

@Override

    public Collection<Item> getItems() {

        return map.values();

    }

Phương thức trả về tất cả các Item

@Override

    public int getCount() {

        int count = 0;

        for (Item item : map.values()) {

            count += item.getQty();

        }

        return count;

    }

Phương thức đếm tổng số lượng Item có trong giỏ hàng

@Override

    public double getAmount() {

        double amount = 0.0;

        for (Item item : map.values()) {

            amount += item.getPrice() \* item.getQty();

        }

        return amount;

    }

Phương thức tính tiền của 1 item dựa vào công thức:Tổng tiền=tiền của item\*số lượng của item

Bài 5 (2 điểm)

-Thực hiện code sản phẩm

Lớp DB:

public static Map<Integer, Item> items = new HashMap<>();

    static {

        items.put(1, new Item(1, "Samsung", 10.0, 0));

        items.put(2, new Item(2, "Nokia 2021", 20.50, 0));

        items.put(3, new Item(3, "iPhone 20", 90.49, 0));

        items.put(4, new Item(4, "Motorola", 15.55, 0));

        items.put(5, new Item(5, "Seamen", 8.99, 0));

    }

@Controller

public class ItemController {

    @RequestMapping("/item/index")

    public String list(Model model) {

        model.addAttribute("items", DB.items.values());

        return "item/index";

    }

}

Controller Item hiển thị toàn bộ sản phẩm

<ul>

        <li th:each="item : ${items}"><span th:text="${item.name}"></span>

            (<span th:text="${item.price}"></span>) :: [<a

            th:href="@{'/cart/add/' + ${item.id}}">Add To Cart</a>]</li>

    </ul>

File index.html trong thư mục item hiển thị lên form sản phẩm

-Thực hiện code giỏ hàng:

@Autowired

    ShoppingCartService cart; // tiêm Spring Bean đã viết ở bài trước

@RequestMapping("/cart/view")

    public String view(Model model) {

        model.addAttribute("cart", cart);

        return "cart/index";

    }

Hiển thị form index trong thư mục cart và thêm model cart

@RequestMapping("/cart/add/{id}")

    public String add(@PathVariable("id") Integer id) {

        cart.add(id);

        return "redirect:/cart/view"; // hiển thị giỏ hàng

    }

Thêm sản phẩm từ sản phẩm vào giỏ hàng

@RequestMapping("/cart/remove/{id}")

    public String remove(@PathVariable("id") Integer id) {

        cart.remove(id);

        return "redirect:/cart/view";

    }

Xóa sản phẩm ra khỏi giỏ hàng

@RequestMapping("/cart/update/{id}")

    public String update(@PathVariable("id") Integer id, @RequestParam("qty") Integer qty) {

        cart.update(id, qty);

        return "redirect:/cart/view";

    }

Cập nhật số lượng sản phẩm

@RequestMapping("/cart/clear")

    public String clear() {

        cart.clear();

        return "redirect:/cart/view";

    }

Xóa toàn bộ sản phẩm trong giỏ hàng

File html index trong thư mục cart:

<table border="1" style="width: 100%">

        <tr>

            <th>Id</th>

            <th>Name</th>

            <th>Price</th>

            <th>Quantity</th>

            <th>Amount</th>

            <th></th>

        </tr>

        <th:block th:each="item : ${cart.items}">

            <form action="/cart/update/{id}" method="post"

                th:action="@{'/cart/update/' + ${item.id}}">

                <input type="hidden" name="id" th:value="${item.id}">

                <tr>

                    <td th:text="${item.id}"></td>

                    <td th:text="${item.name}"></td>

                    <td th:text="${item.price}"></td>

                    <td><input name="qty" th:value="${item.qty}"

                        onblur="this.form.submit()" style="width: 50px;"></td>

                    <td th:text="${item.price \* item.qty}"></td>

                    <td><a th:href="@{'/cart/remove/' + ${item.id}}">Remove</a></td>

                </tr>

            </form>

        </th:block>

    </table>

    <a href="/cart/clear">Clear Cart</a>

    <a href="/item/index">Add more</a>