

SpringerBriefs in Computer Science

Pedro Mejia Alvarez · Raul E. Gonzalez Torres ·
Susana Ortega Cisneros

Exception Handling

Fundamentals and Programming



Springer

SpringerBriefs in Computer Science

SpringerBriefs present concise summaries of cutting-edge research and practical applications across a wide spectrum of fields. Featuring compact volumes of 50 to 125 pages, the series covers a range of content from professional to academic.

Typical topics might include:

- A timely report of state-of-the art analytical techniques
- A bridge between new research results, as published in journal articles, and a contextual literature review
- A snapshot of a hot or emerging topic
- An in-depth case study or clinical example
- A presentation of core concepts that students must understand in order to make independent contributions

Briefs allow authors to present their ideas and readers to absorb them with minimal time investment. Briefs will be published as part of Springer's eBook collection, with millions of users worldwide. In addition, Briefs will be available for individual print and electronic purchase. Briefs are characterized by fast, global electronic dissemination, standard publishing contracts, easy-to-use manuscript preparation and formatting guidelines, and expedited production schedules. We aim for publication 8–12 weeks after acceptance. Both solicited and unsolicited manuscripts are considered for publication in this series.

****Indexing:** This series is indexed in Scopus, Ei-Compendex, and zbMATH **

Pedro Mejia Alvarez • Raul E. Gonzalez Torres
Susana Ortega Cisneros

Exception Handling

Fundamentals and Programming



Springer

Pedro Mejia Alvarez
CINVESTAV-Guadalajara
Zapopan, Jalisco, Mexico

Raul E. Gonzalez Torres
CINVESTAV-Guadalajara
Zapopan, Jalisco, Mexico

Susana Ortega Cisneros
CINVESTAV-Guadalajara
Zapopan, Jalisco, Mexico

ISSN 2191-5768

ISSN 2191-5776 (electronic)

SpringerBriefs in Computer Science

ISBN 978-3-031-50680-2

ISBN 978-3-031-50681-9 (eBook)

<https://doi.org/10.1007/978-3-031-50681-9>

© The Editor(s) (if applicable) and The Author(s), under exclusive license to Springer Nature Switzerland AG 2024

This work is subject to copyright. All rights are solely and exclusively licensed by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors, and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, expressed or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Paper in this product is recyclable.

Preface

Whether you're building a simple app or an intricate software system, things can go wrong. These unexpected events, or "exceptions", can range from trivial matters like a missing file to severe errors that can crash an entire system. If these exceptions are not handled, they can lead to unreliable and unpredictable software behavior. Exception handling is the process of responding to the occurrence of exceptions – abnormal or exceptional conditions requiring special processing – during the software's execution. Exception handling is crucial in producing robust software that can cope with unexpected situations, ensuring the software is more resilient, maintainable, and user-friendly.

In this book, we will delve deep into the world of exception handling with examples written in C++ and Python. Starting with its history and evolution, we will explore the many facets of exception handling, such as its syntax, semantics, challenges, best practices, and more. You'll understand the nuances between syntax and semantic errors, learn how to employ try-catch blocks effectively, grasp the importance of logging exceptions, and even delve into advanced exception-handling techniques.

The following chapters will provide you with a comprehensive understanding of this crucial software development concept:

Chapter 1 provides an introduction, covering the history, various definitions, and challenges of exception handling. Chapter 2 delves into the basics, offering insights into the foundational concepts and techniques. Chapter 3 touches upon the best practices for exception handling, including the differences between errors and exceptions, the use of assertions, and how to provide meaningful error messages. Chapter 4 takes a deep dive into advanced exception-handling techniques. Here, we explore patterns, guard clauses, hierarchical exception handling, and much more. Finally, chapter 5 focuses on the complexities of exception handling in real-time and embedded systems.

Whether you are a seasoned developer looking to refine your understanding of exception handling or a newcomer eager to grasp its essentials, this book offers a clear, thorough, and practical guide to mastering this crucial area of software development. As you progress through the chapters, you will acquire the knowledge and skills needed to handle exceptions effectively, ensuring that your software applications

are more robust, reliable, and resilient to unexpected disruptions. Welcome to the enlightening journey of mastering exception handling!

CINVESTAV-Guadalajara, Mexico
CINVESTAV-Guadalajara, Mexico
CINVESTAV-Guadalajara, Mexico

Pedro Mejia-Alvarez
Raul. E. Gonzalez-Torres
Susana Ortega-Cisneros

Contents

1	Introduction to Exception Handling	1
1.1	History and Evolution of Exception Handling	1
1.2	Definition of Exceptions	2
1.2.1	Examples of Exceptions	3
1.2.2	Exception Objects and Structures	5
1.2.3	Checked and Unchecked Exceptions	7
1.2.4	Internal and External Exception Handling	8
1.2.5	Attachment of Handlers	10
1.3	Challenges of Exception Handling	12
1.3.1	Preventing Crashes and Unexpected Behavior	12
1.3.2	Improving Robustness and Maintainability	13
1.3.3	Enhancing Security	13
1.4	Syntax and Semantic Errors	14
1.4.1	Syntax Errors	14
1.4.2	Semantic Errors	14
1.5	Design by Contract: A Precursor to Exception Handling	15
1.5.1	Principles of Design by Contract	15
1.5.2	Relationship with Exceptional Situations	15
2	Basics of Exception Handling	17
2.1	Try-Catch Blocks	17
2.2	The Throw Keyword in Exception Handling	19
2.3	Custom Exceptions Handling	19
2.4	Exception Propagation in Exception Handling	21
2.5	Nested Try-Catch Blocks	23
2.5.1	Benefits of Nested try-catch blocks	23
2.5.2	Challenges of Nested Try-Catch Blocks	24
2.5.3	Examples	24
2.5.4	Best Practices	25
2.6	Conditional Exception Handling	25
2.6.1	Event-driven Exception Handling	26

3	Exception Handling Best Practices	29
3.1	Differences between Errors and Exceptions	30
3.2	Errors as Undesired Events	31
3.2.1	Critical Aspects to Undesired Events	31
3.3	Error Codes and Return Values in Exception Handling	33
3.3.1	Error Codes in C++	33
3.3.2	Return Values as Errors	34
3.4	Assertions and Exception Handling	35
3.4.1	Assertions	35
3.4.2	Assertions vs. Exception Handling	35
3.5	Using Specific Exception Types	36
3.5.1	Examples of Specific Exception Types	38
3.6	Providing Meaningful Error Messages	39
3.6.1	Examples of Meaningful Error Messages	41
3.7	Logging Exceptions	41
3.7.1	Objectives	41
3.7.2	Challenges	42
3.7.3	Examples of Logging Exceptions	42
3.8	Handling Exceptions at the Appropriate Level	44
3.9	Implementing Graceful Degradation and Recovery Strategies	46
3.10	Employing Separation of Concerns in Exception Handling	46
3.11	Avoiding Empty Catch Blocks	47
4	Advanced Exception Handling Techniques	49
4.1	Exception Handling Patterns	49
4.2	Guard Clauses	50
4.2.1	Exception Translation	51
4.2.2	Exception Aggregation	52
4.2.3	Exception Shielding Pattern	55
4.3	Retry and Backoff Strategies for Exception Handling	58
4.3.1	Fixed Interval Retry	58
4.3.2	Randomized Interval Retry	59
4.3.3	Decorrelated Jitter Backoff	60
4.4	Exception Handling in Multi-Threaded Environments	60
4.4.1	Techniques for Exception Handling in Multi-Threaded Environments	61
4.4.2	Challenges and Limitations	63
4.5	Hierarchical Exception Handling	64
4.5.1	Default Exception Handling	65
4.5.2	Customizing the Hierarchy	66
4.5.3	Testing and Maintaining the Hierarchy	66
4.5.4	Exception Propagation	66
4.5.5	Hierarchical Exception Handling in Different Programming Paradigms	67
4.5.6	Benefits and drawbacks of Hierarchical Exception Handling	68

- 4.6 Clean-Up Actions in Exception Handling 70
 - 4.6.1 The Necessity of Clean-Up Actions 70
 - 4.6.2 Clean-Up Techniques in Different Programming Languages . 71
 - 4.6.3 Best Practices for Clean-Up Actions 72
- 5 Exception Handling in Real-Time and Embedded Systems 73**
 - 5.1 Example of Exceptions in Real-Time Systems 74
 - 5.2 Constraints in Real-Time Systems 75
 - 5.2.1 Timing Constraints 75
 - 5.2.2 Resource Constraints 76
 - 5.2.3 Predictability and Determinism 76
 - 5.2.4 Reliability and Fault Tolerance 77
 - 5.3 Types of Timing Exceptions in Real-time Systems 77
 - 5.3.1 Hardware Timing Exceptions 78
 - 5.3.2 Software Timing Exceptions 79
 - 5.4 Priorities and Deadlines in Exceptions Handling in Real-time Systems 80
 - 5.4.1 Prioritizing Exceptions 81
 - 5.4.2 Deadline Management 81
 - 5.5 Interrupts and Exception Handling in C++ 83
 - 5.5.1 Linking Interrupts and Exceptions 84
 - 5.5.2 Similarities and differences of Interrupts and Exceptions 85
 - 5.5.3 Combining Interrupts and Exceptions 86
 - 5.6 Methodologies for Exception Handling in Real-Time Systems 87
 - 5.6.1 Recovery Blocks 88
 - 5.6.2 Checkpoints and Rollbacks 89
 - 5.6.3 Proactive Monitoring of Exception Handling in Real-Time Systems 89
 - 5.7 Design Patterns for Exception Handling in Real-Time Systems 91
 - 5.7.1 The Error Handler Pattern in Real-Time Systems 92
 - 5.7.2 State Machine Pattern in Real-Time Systems 94
 - 5.7.3 Supervisory Control Pattern 96
 - 5.8 Exception Handling in Ada for Real-Time Systems 98
 - 5.8.1 Defined Exception Model in Ada 99
 - 5.8.2 Deterministic Exception Propagation in Ada 100
 - 5.8.3 Explicit Exception Declarations in Ada 102
 - 5.8.4 Hierarchical and Scoped Handling in Ada 103
 - 5.8.5 Tasking and Exception Handling in Ada 105
 - 5.8.6 Real-time Considerations in Handler Actions 107
- References 110**



Chapter 1

Introduction to Exception Handling

Exception handling is an essential aspect of software development that enables programmers to manage unexpected events and errors during program execution.

The chapter begins by providing a general overview of the history and evolution of exception handling. Then we describe key concepts and principles of exception handling, including the types of exceptions, how they are defined and raised, and the different approaches to handling them. This section also delineates the definition of exceptions, that navigate through the architecture of exception objects, distinctions between checked and unchecked exceptions, and strategies for internal and external handling. Furthermore, the process of attaching handlers to specific exceptions is spotlighted, emphasizing its pivotal role in effective exception management. Exception handling is not just a theoretical construct; its implications are profound and far-reaching. In the next section, the chapter underscores the paramount importance of adept exception handling. By averting crashes and unexpected behavior, fostering robustness and maintainability, and bolstering security, exception handling elevates software quality, ensuring users experience reliability and trust in their interactions with systems. Finally, we also delve into the different types of exceptions, including syntax and semantic errors.

By the end of the chapter, readers will have a solid understanding of the key concepts and principles of exception handling and the tools and frameworks available for implementing effective exception handling in their software projects. The chapter sets the foundation for the rest of the book, which details specific topics and techniques related to exception handling in software development.

1.1 History and Evolution of Exception Handling

The history of exception handling [67] can be traced back to the early days of programming, when languages like Assembly and Fortran were the primary means of coding. As programming languages and paradigms advanced, the need for better error and exception-handling mechanisms became apparent. This section provides an extensive overview of the evolution of exception handling throughout the years. In the

1950s and 1960s, assembly language and Fortran were the dominant programming languages. They relied on fundamental error-checking mechanisms, using simple branching instructions to handle errors. As a result, programmers have to write extensive code to handle different types of errors and exceptions, often leading to less maintainable and more error-prone code.

One of the first programming languages to introduce structured exception handling was PL/I in the 1960s. The language featured condition handling, where specific conditions could be associated with particular handlers. However, the system still required manual management of error propagation, making it less flexible and more error-prone than modern exception-handling systems. The advent of object-oriented programming languages like C++ and Python played a crucial role in popularizing the try-catch-finally construct. This construct allowed developers to write more structured, maintainable code when dealing with exceptions. In addition, the exception-handling mechanism provided a clear separation of concerns, with specific blocks of code designated for error handling and recovery.

As programming languages continue to evolve, newer languages have adopted more advanced and flexible exception-handling constructs. For example, languages like Python, C#, and Ruby support exception-handling constructs similar to those found in Java, focusing on making the code more readable and maintainable. Scripting languages, such as JavaScript and PHP, also feature their exception-handling mechanisms. For example, while JavaScript relies on the try-catch-finally construct, similar to Java and C++, PHP uses try-catch with optional finally blocks.

In conclusion, exception handling has come a long way since the early days of programming, with each programming paradigm introducing new approaches and constructs to handle errors and exceptions effectively. Therefore, studying the history and evolution of exception-handling mechanisms provides valuable insights into the progress of programming languages and the software development process.

1.2 Definition of Exceptions

An exception can be defined as an unexpected event or condition that occurs during the execution of a program, leading to abnormal or undesired behavior [31]. These events or conditions deviate from the expected or regular flow of the program. They may result in various consequences, such as incorrect outputs, system crashes, or data corruption. Exception handling is the process of managing and resolving exceptions to allow the program to continue executing or terminate gracefully without causing further harm.

Exceptions can arise from various sources, such as:

- Errors in the code or logic of the program
- Invalid user inputs or actions
- Resource limitations or constraints, such as running out of memory or disk space
- Hardware failures or malfunctions
- External factors, such as network interruptions or loss of communication with remote services