**Policy Management System for Mashfords Insurance Company**

A comprehensive Python-based insurance management system that handles policyholders, products, policies, and payments with automated processing and tracking capabilities.

Table of Contents

[🚀 Features 2](#_Toc206963329)

[Core Functionality 2](#_Toc206963330)

[Advanced Features 3](#_Toc206963331)

[🏗️ System Architecture 3](#_Toc206963332)

[Status Enumerations 3](#_Toc206963333)

[📦 Installation 3](#_Toc206963334)

[Prerequisites 3](#_Toc206963335)

[Setup 3](#_Toc206963336)

[💻 Usage 3](#_Toc206963337)

[Basic Setup 3](#_Toc206963338)

[Running the Demo 3](#_Toc206963339)

[📚 Class Documentation 3](#_Toc206963340)

[Policyholder 3](#_Toc206963341)

[Key Methods: 3](#_Toc206963342)

[Product 4](#_Toc206963343)

[Key Methods: 4](#_Toc206963344)

[Policy 4](#_Toc206963345)

[Key Methods: 4](#_Toc206963346)

[Payment 4](#_Toc206963347)

[Key Methods: 4](#_Toc206963348)

[InsuranceManager 4](#_Toc206963349)

[Key Methods: 4](#_Toc206963350)

[🔄 System Workflow 5](#_Toc206963351)

[1. Initial Setup 5](#_Toc206963352)

[2. Create Products 5](#_Toc206963353)

[3. Register Policyholders 5](#_Toc206963354)

[4. Create Policies 5](#_Toc206963355)

[5. Generate Payments 5](#_Toc206963356)

[6. Process Payments 5](#_Toc206963357)

[7. Handle Overdue Payments 5](#_Toc206963358)

[🛡️ Error Handling 5](#_Toc206963359)

[Validation Errors 5](#_Toc206963360)

[Business Logic Errors 5](#_Toc206963361)

[Example Error Handling 5](#_Toc206963362)

[📊 Example Output 6](#_Toc206963363)

[=== INSURANCE SYSTEM SUMMARY === 6](#_Toc206963364)

[🤝 Contributing 6](#_Toc206963365)

[Development Guidelines 7](#_Toc206963366)

[📝 License 7](#_Toc206963367)

[🏢 About Mashfords Insurance Company 7](#_Toc206963368)

[📞 Support 7](#_Toc206963369)

# 🚀

# Features

## Core Functionality

* **Policyholder Management**: Create, suspend, reactivate, and deactivate policyholders
* **Product Management**: Manage insurance products with pricing and status controls
* **Policy Lifecycle**: Create, renew, and cancel insurance policies
* **Payment Processing**: Handle payments with due dates, penalties, and status tracking
* **Automated Overdue Processing**: Automatically detect and process overdue payments
* **System Analytics**: Generate comprehensive system summaries and reports

## Advanced Features

* **Enum-based Status Management**: Type-safe status handling for all entities
* **Comprehensive Validation**: Input validation with detailed error messages
* **Relationship Management**: Track relationships between policyholders, policies, and products
* **Penalty System**: Automatic penalty calculation and application for late payments
* **Date-based Operations**: Time-sensitive operations with proper datetime handling

# 🏗️ System Architecture

The system follows object-oriented principles with the following main components:

InsuranceManager (Central Controller)

├── Policyholder (Customer Management)

├── Product (Insurance Products)

├── Policy (Policy Lifecycle)

└── Payment (Payment Processing)

## Status Enumerations

* **PolicyholderStatus**: ACTIVE, SUSPENDED, INACTIVE
* **ProductStatus**: ACTIVE, SUSPENDED, DISCONTINUED
* **PaymentStatus**: PENDING, PAID, OVERDUE, LATE, PENALTY\_APPLIED

## 📦 Installation

### Prerequisites

* Python 3.7 or higher
* No external dependencies required (uses only Python standard library)

### Setup

1. Clone or download the repository
2. Ensure Python 3.7+ is installed on your system
3. Run the system:

python insurance\_management.py

# 💻 Usage

## Basic Setup

from insurance\_management import \*

# Create insurance manager

manager = InsuranceManager()

# Create products

health\_insurance = Product(101, "Health Insurance", 500.0)

manager.add\_product(health\_insurance)

# Create policyholders

john\_doe = Policyholder(1, "John Doe")

manager.add\_policyholder(john\_doe)

# Create policies

policy = manager.create\_policy(1001, 1, 101)

# Create and process payments

payment = manager.create\_payment(2001, 1001)

payment.process\_payment()

## Running the Demo

The system includes a comprehensive demonstration that showcases all features:

python insurance\_management.py

This will run through a complete workflow including:

* Setting up products and policyholders
* Creating policies
* Processing payments
* Demonstrating management features
* Handling overdue payments
* Generating system reports

# 📚 Class Documentation

## Policyholder

Manages customer information and status.

### Key Methods:

* suspend(reason=""): Suspend a policyholder with optional reason
* reactivate(): Reactivate a suspended policyholder
* is\_active(): Check if policyholder is active
* get\_active\_policies(): Get all active policies for this policyholder

### Product

Manages insurance products and pricing.

### Key Methods:

* update(name=None, premium=None): Update product details
* suspend(reason=""): Suspend a product
* reactivate(): Reactivate a product
* is\_active(): Check if product is active

# Policy

Manages the relationship between policyholders and products.

## Key Methods:

* cancel(): Cancel the policy
* renew(duration\_days=365): Renew policy for specified duration
* is\_active(): Check if policy is currently active

## Payment

Handles payment processing and tracking.

## Key Methods:

* process\_payment(payment\_amount=None): Process a payment
* send\_reminder(): Send payment reminder
* check\_overdue(): Check and update overdue status
* apply\_penalty(penalty\_rate=0.1): Apply penalty for late payment

# InsuranceManager

Central management system for all entities.

## Key Methods:

* create\_policy(policy\_id, policyholder\_id, product\_id): Create new policy
* create\_payment(payment\_id, policy\_id, amount=None): Create new payment
* process\_overdue\_payments(): Process all overdue payments
* get\_summary(): Display system summary

# 🔄 System Workflow

## 1. Initial Setup

manager = InsuranceManager()

## 2. Create Products

product = Product(101, "Health Insurance", 500.0)

manager.add\_product(product)

## 3. Register Policyholders

holder = Policyholder(1, "John Doe")

manager.add\_policyholder(holder)

## 4. Create Policies

policy = manager.create\_policy(1001, 1, 101)

## 5. Generate Payments

payment = manager.create\_payment(2001, 1001)

## 6. Process Payments

payment.process\_payment()

## 7. Handle Overdue Payments

manager.process\_overdue\_payments()

# 🛡️ Error Handling

The system includes comprehensive error handling:

## Validation Errors

* **Invalid IDs**: Must be positive integers
* **Empty Names**: Names cannot be empty or whitespace-only
* **Negative Premiums**: Premium amounts must be non-negative
* **Invalid Relationships**: Policies require valid policyholders and products

## Business Logic Errors

* **Inactive Entity Operations**: Cannot create policies with inactive entities
* **Payment Processing**: Prevents processing already-paid payments
* **Policy Renewal**: Cannot renew cancelled policies

## Example Error Handling

try:

product = Product(-1, "Invalid Product", 100.0)

except ValueError as e:

print(f"Error: {e}") # Output: Error: Product ID must be a positive integer

# 📊 Example Output

When running the demo, you'll see output like:

=== Setting up Insurance Management System ===

Product Health Insurance added to system.

Product Car Insurance added to system.

Product Life Insurance added to system.

Policyholder Bezzel Chitsungo added to system.

Policyholder Shellah Murwira added to system.

Policyholder Mukudzei Zuruvi added to system.

=== Creating Policies ===

Policy 1001 created for Bezzel Chitsungo with product Health Insurance.

Policy 1002 created for Shellah Murwira with product Health Insurance.

...

# === INSURANCE SYSTEM SUMMARY ===

Active Policyholders: 3/3

Active Products: 3/3

Active Policies: 4/4

Pending Payments: 1/4

================================

## 🤝 Contributing

1. Fork the repository
2. Create a feature branch (git checkout -b feature/new-feature)
3. Commit your changes (git commit -am 'Add new feature')
4. Push to the branch (git push origin feature/new-feature)
5. Create a Pull Request

## Development Guidelines

* Follow PEP 8 style guidelines
* Add type hints for all new methods
* Include comprehensive error handling
* Add docstrings for new classes and methods
* Test all new functionality

# 📝 License

This project is licensed under the MIT License - see the [LICENSE](https://claude.ai/chat/LICENSE) file for details.

# 🏢 About Mashfords Insurance Company

This system was developed as Milestone Assignment 1 for Mashfords Insurance Company's policy management requirements. It demonstrates best practices in Python development, object-oriented design, and insurance domain modeling.

# 📞 Support

For support, please contact the development team or raise an issue in the repository.

Team Leader: Dr Eng. B Chitsungo, eng.chitsungo@gmail.com

**Version**: 1.0.0  
**Last Updated**: August 2025  
**Python Compatibility**: 3.7+