Error Handling and Failing Gracefully

Sometimes things happen when the program runs that the program has no control over. Files will not exist, or the program may not have permission to access them. Network connections fail. Users will enter bad data. When these things happen, the program must respond correctly.

There are two things a program can do when these errors arise. The program can try to fix the problem, or the program can shut down. Fixing the problem is the preferred option but cannot always be done. Shutting down must be done in a user-friendly way; we call this "failing gracefully".

Always remember that the primary threat to any computing system is the authorized user. An authorized user already has permission to access things. Even a well-meaning user can make mistakes that cause problems with the systems.

Fixing the Problem

When possible, the control logic of the program should try to fix the problem. For example, if the problem is a lost network connection, the program may try to reconnect. If the user has entered invalid data, the program may ask the user to try again.

> A program is considered ***robust*** when it can handle errors and exceptions and continue to run correctly. Programmers make programs robust by assuming that things will go wrong and writing code to handle them.

Consider a business with a shipping department. What should happen if the shipping company (the company that actually transports the packages) has neglected to pick up the outgoing shipments? Who should handle that and what should they do?

If the shipping department manager calls the shipping company, that may resolve the issue and get the packages picked up. But does that solve all the issues? Will packages be late?

If there are other issues, such as the need to call customers about late shipments, then the shipping manager is not the person to handle the issue. Instead, the issue needs to be bumped up to a higher-level manager who can coordinate the solution. That manager may delegate the necessary tasks to others. The customer service people may be instructed to call customers. The shipping manager may be instructed to call the shipping company. The manager may check to see if this is a contract violation with the shipping company to see if there are any other required actions.

Similarly, your must determine which part of your program must handle exceptions. A function that attempts to read a file should not communicate directly with the user because that would be too many responsibilities. If there is a file error, the file handling function should not respond. Instead, the control logic should decide what to do, including telling user interface functions to let the user know what happened.

Deciding where and how to handle exceptions takes experience, so don't worry if you don't get it right immediately. With practice, you will improve the efficiency.

Failing Gracefully

In some cases, there is nothing that can be done about the exception. In such a case, the program needs to shut down. This can be stressful for some users, so care is needed.

Remember the hypothetical user, who is skittish and irritable. If your program crashes, the user will be upset. The user may be worried that they will be blamed for whatever happened. If your program makes them uncomfortable, they will not want to use it no matter how well your code was made.

To help with this, we have the concept of failing gracefully. This is when the program crashes but does so in a friendly way that looks like it is intentional and correct. The program still shuts down and still gives error messages, but they are friendly and controlled.

When the program fails, the communication needs to do several things.

1. Tell the user that the program is ending.
2. Tell the user why the program is ending in a user-friendly, non-technical way.
3. Tell the user what they need to do next.

When telling the user why the program is ending, it is important for the user to feel that it is not because they did something wrong (even if they did). The program is not blaming them; it is just explaining the issue. At the same time, the error message must be clear but understandable to the user. You will need to determine the likely user's level of technical understanding; but assume less knowledge.

One of the real keys is telling the user what to do next. When something goes wrong for a non-technical user, it can be very confusing. They may not understand what happened or what they should do about it. If your program can tell the user what to do, it will help them feel much better.

Some of the things you may tell the user might include:

- Restart the program and try again.
- Check on a missing resource such as a file or network item.
- Contact the technical support people and pass along an error code.

If the user knows what to do and why to do it, they will feel more comfortable and will be ready to move on with their work.

Always organize your program to fail gracefully!

If your program does not fail gracefully when something goes wrong, then your program has crashed. If your program crashes, even due to things beyond the control of your program, the users will think your program is bad and will not trust it. They also believe that bad programs are written by bad programmers.

Conclusion

Anything outside of your program can cause exceptions. Files, permissions, network connections, and user inputs are all outside of your control. The biggest threat to all the computing systems is the authorized users.

You must make your code as robust as possible and prepare for exceptions. Exceptions must be handled by the part of the program that can make the best decision. In all cases, the program should handle the exceptions and not crash.

Handling exceptions comes with experience. Over time, you will get better at handling exceptions.